НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”
ІМЕНІ ІГОРЯ СІКОРСЬКОГО
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ
КАФЕДРА СИСТЕМНОГО ПРОГРАМУВАННЯ
І СПЕЦІАЛІЗОВАНИХ КОМ’ЮТЕРНИХ СИСТЕМ

«До захисту допущено»
Завідувач кафедри
_____ В.П. Тарасенко
(підпис)
“___”_____2019 р.

Дипломна робота

освітньо-кваліфікаційного рівня “Бакалавр”

з напряму підготовки 6.050102 “Комп’ютерна інженерія”

на тему МУРАШИНИЙ АЛГОРИТМ З МЕТАФОРОЮ АГРЕГАЦІЇ ФЕРОМОНІВ ДЛЯ ГЛОБАЛЬНОЇ ОПТИМІЗАЦІЇ

Виконав: студент 4 курсу, групи КВ-52
Абдураімов Таір Заірович _____
(підпис)

Керівник доц., к.т.н. Зорін Ю.М. _____
(підпис)

Рецензент _____ _____
(підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019

Національний технічний університет України

“Київський політехнічний інститут” імені Ігоря Сікорського

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп’ютерних систем

Освітньо-кваліфікаційний рівень “Бакалавр”

Напрям підготовки 6.050102 “Комп’ютерна інженерія”

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ В.П. Тарасенко

“___” _____

2019 р.

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Абдураімову Таіру Заіровичу

1. Тема роботи: МУРАШИНИЙ АЛГОРИТМ З МЕТАФОРОЮ АГРЕГАЦІЇ ФЕРОМОНІВ ДЛЯ ГЛОБАЛЬНОЇ ОПТИМІЗАЦІЇ,
керівник роботи: Зорін Юрій Михайлович, к.т.н., доцент,
затверджені наказом по університету від “__” _____ 2019 року № ____.

2. Строк подання студентом роботи: “__” червня 2019 р.

3. Вихідні дані для дипломної роботи:
   – оригінальний мурашиний алгоритм для глобальної оптимізації в неперервному просторі.

4. Перелік задач, які потрібно вирішити:
   – вивчити літературні джерела за тематикою дослідження, в тому числі методики оптимізації в непе

- ервному просторі;
- провести аналіз алгоритмів, які використовуються для оптимізації в неперервному просторі;
- обґрунтувати вибір методу оптимізації;
- розробити комп'ютерну модель алгоритму;
- оцінити переваги та недоліки алгоритму.

5. Перелік обов'язкового ілюстративного матеріалу:
   - схема мурашиного алгоритму оптимізації;
   - схема мурашиного алгоритму оптимізації з метафорою агрегації феромонів;
   - порівняльні таблиці.

6. Дата видачі завдання: "__" _____ 2019 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Строк виконання етапів	Примітка
1.	Вивчення літератури за тематикою роботи та збір даних	11.01.2019	
2.	Розроблення та узгодження завдання	01.02.2019	
3.	Аналіз існуючих алгоритмів оптимізації	13.03.2019	
4.	Підготовка матеріалів першого розділу дипломної роботи	31.03.2019	
5.	Підготовка матеріалів другого розділу дипломної роботи	09.04.2019	
6.	Розробка і тестування комп'ютерної програми	20.04.2019	
7.	Підготовка матеріалів третього розділу дипломної роботи	29.04.2019	
8.	Підготовка матеріалів четвертого розділу дипломної роботи	10.05.2019	
9.	Підготовка графічної частини дипломної роботи	20.05.2019	
10.	Оформлення дипломної роботи	28.05.2019	

Студент _____ Абдураімов Т. З.

Керівник роботи _____ Зорін Ю. М.

АНОТАЦІЯ

Кваліфікаційна робота включає пояснювальну записку (56 с., 14 рис. 21 табл., 2 додатки).

Об’єкт розробки – процес оптимізації дійсної функції багатьох змінних в неперервному просторі. Метою роботи є розробка алгоритму оптимізації мурашиної колонії з метафорою агрегації феромонів для пошуку екстремумів дійсної функції.

Запропоновано модифікацію мурашиного алгоритму оптимізації в неперервному просторі у вигляді системи агрегації феромонів з метою покращення точності й сталості результатів. Виконано порівняльний аналіз алгоритму з класичним мурашиним а також з іншими евристичними алгоритмами, які оптимізовані для розв’язку задач в неперервному просторі.

Проведена імплементація розробленого алгоритму для деяких відомих тестових функцій. Здійснена на мові програмування C++. Були визначені параметри алгоритму, знайдені оптимальні їх значення.

На основі аналізу розробленого алгоритму зроблені висновки, визначені його основні переваги і недоліки.

Ключові слова:

ЗАДАЧА ОПТИМІЗАЦІЇ, АЛГОРИТМ ОПТИМІЗАЦІЇ МУРАШИНОЇ КОЛОНІЇ, АГРЕГАЦІЯ ФЕРОМОНІВ, ЕВРИСТИЧНИЙ АЛГОРИТМ, ТЕСТОВІ ФУНКЦІЇ, C++.

SUMMARY

Qualifying work includes explanatory note.

The object of development - the process of optimizing the real function of many variables in a continuous space. The aim of the work is to develop an ant colony optimization algorithm with a pheromone aggregation metaphor to find the extremums of a real function.

The modification of the ant algorithm optimization in the continuous space with a system of pheromones aggregation is proposed in order to improve the accuracy and stability of the results. The comparative analysis of the algorithm with classical ant as well as other heuristic algorithms optimized for solving problems in a continuous space is performed.

Implementation of the developed algorithm for some known test functions has been carried out. Made in the programming language C ++. The parameters of the algorithm were determined and their optimal values were found.

Based on the analysis of the developed algorithm, conclusions are drawn, its main advantages and disadvantages are determined.

Keywords:

THE PROBLEM OF OPTIMIZATION, THE ANT COLONY OPTIMIZATION ALGORITHM, THE PHEROMONE AGREGATION, HEURISTIC ALGORITHM, TEST FUNCTIONS, C ++.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

МАО – мурашиний алгоритм оптимізації;

МАНП – мурашиний алгоритм оптимізації в неперервному просторі;

ЩЙ – функція щільності ймовірностей (розподіл Гауса);

ДО – дослідження операцій;

МРЧ – метод рою часток;

АБК – алгоритм бджолиної колонії;

НПП –найкраща персональна позиція;

НГП – найкраща глобальна позиція;

ТП – алгоритм табу-пошуку;

АІВ – алгоритм імітації відпалу;

ГА – генетичний алгоритм;

ДЕ – диференційна еволюція;

САФ – система агрегації феромонів;

RFID (Radio Frequency Identification) – радіочастотна ідентифікація;

GPRS (General Packet Radio Service) - пакетний радіозв’язок загального використання;

ВСТУП

Оптимізація – це процес знаходження найвигіднішого варіанту. Людина по своїй природі завжди прагне знайти краще рішення в будь-якій проблемі. Можна виділити такі відомі проблеми оптимізації, як задача комівояжера та задача пакування рюкзака. В сучасному світі однією з найважливіших проблем є питання логістики, тобто керування матеріальними, інформаційними і людськими ресурсами з метою їх оптимізації.

В математиці задача оптимізації полягає в знаходженні екстремумів заданої функції в неперервному просторі. Для розв'язку такої задачі існує багато методів, в тому числі для оптимізації використовують евристичні алгоритми (евристики) – це алгоритми, що включають практичний метод, який не є точним або оптимальним, але достатнім для розв'язку поставленої задачі.

В даній дипломній роботі був розроблений алгоритм оптимізації на основі алгоритму мурашиної колонії. Мурашиний алгоритм – евристичний алгоритм для знаходження наближених розв'язків задачі комівояжера, а також аналогічних завдань пошуку маршрутів на графах. Підхід запропонований бельгійським дослідником Марко Доріго. Суть підходу полягає в аналізі та використанні моделі поведінки мурах, що шукають дороги від колонії до їжі.

Отже, на основі мурашиного алгоритму був розроблений алгоритм оптимізації в неперервному просторі з використанням системи агрегації феромонів. Представлена програмна реалізація алгоритму, яка була протестована на різних функціях, виділені його основні переваги і недоліки в порівнянні з іншими відомими евристиками, такими як генетичний алгоритм.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ І ОБГРУНТУВАННЯ ТЕМИ ДИМЛОМНОЇ РОБОТИ

1.1. Постановка задачі оптимізації

Як відомо, екстремум функції – це найбільше чи найменше її значення на заданій множині. Розрізняють локальні та глобальні екстремуми. Локальний – це екстремум в деякому достатньо малому околі даної точки, глобальний – в усій розглядуваній області значень. Задача оптимізації – це знаходження точки (або точок) глобального екстремуму або декількох екстремумів заданої функції.

Отже, задача оптимізації полягає в наступному. Нехай задано функцію f(x), визначену на множині X із n-вимірного евклідового простору . Необхідно знайти точки мінімуму (або максимуму) значень функції f(x) на X. Тобто:

$$f(x) \to \min,\ x \in X.$$

де f(x) — цільова функція, X — допустима множина, кожна точка x цієї множини — допустима точка задачі.

Для того щоб правильно поставити задачу оптимізації необхідно задати:

1. Допустиму множину — множину $X \subset R_n$.
2. Цільову функцію f(x).
3. Критерій пошуку (максимум або мінімум).

В дипломній роботі цільовими функціями виступають так звані тестові функції для оптимізації (штучні ландшафти) — нелінійні функції, які використовуються для оцінки характеристик алгоритмів оптимізації, таких як:

швидкість збіжності; точність; сталість; загальні характеристики. Прикладами таких функцій є Ackley, Sphere, Rosenbrock.

1.2. Аналіз існуючих рішень

Розглянемо класифікацію методів оптимізації.

Існуючі на цей час алгоритми пошуку можна розділити на такі великі групи:

- детерміновані – оптимізація відбувається за деякою визначеною процедурою;
- випадкові (стохастичні) – використовують випадкові величини при обчислені;
- комбіновані.

За критерієм вимірності допустимої множини, методи оптимізації поділяють на наступні методи:

- одномірної оптимізації;
- багатомірної оптимізації.

За вимогами до гладкості і наявності в цільової функції частинних похідних, їх можна розділити на:

- прямі методи, які вимагають тільки обчислень цільової функції в точках наближень;
- методи першого порядку: вимагають обчислення перших частинних похідних функції, тобто якобіана цільової функції;
- методи другого порядку: вимагають обчислення других частинних похідних, тобто гессіана цільової функції.

Крім того методи оптимізації за принципом роботи поділяються на такі групи:

- аналітичні методи (наприклад, метод множників Лагранжа);
- чисельні методи (евристичні і ітераційні методи);
- графічні методи.

Залежно від природи допустимої множини X задачі оптимізації класифікуються таким чином:

- задачі комбінаторної оптимізації — якщо X скінченна або зліченна;
- задачі цілочислового програмування — X є підмножиною множини цілих чисел;
- задачі нелінійного програмування — цільова функція містить нелінійні функції;
- задача лінійного програмування — цільова функція містить лише лінійні функції.

Предметом розгляду в даній роботі є саме обчислювальні (чисельні) методи оптимізації. Для розв'язання задач, дослідники можуть використовувати алгоритми, які зупиняються за скінченну кількість кроків: або ітераційні методи, які збігаються до рішення (на певному класі задач), або евристики, які можуть надати приблизні рішення деяких задач (хоча їхні ітерації не обов'язково будуть сходитись).

Ітераційні методи, що використовуються для розв’язання задач нелінійного програмування, як будо зазначено вище розрізняються залежно від того, чи оцінюють вони гессіан, градієнт або тільки значення функцій. При оцінці гессіану (H) і градієнту (G) швидкість збіжності покращується для функцій, для яких ці величини існують і змінюються досить гладко, проте використання цих оцінок збільшує обчислювальну складність кожної ітерації. У деяких випадках обчислювальна складність може бути занадто високою.

Одним з головних критеріїв для задачі оптимізації є кількість необхідних оцінок функцій, оскільки це часто потребує набагато більше

обчислень, ніж потрібно самому алгоритму оптимізації, який здебільшого мусить оперувати над N змінними. Також варто відзначити, що хоч і похідні надають детальну інформацію для оптимізаторів, але їх важче обчислити, наприклад, апроксимація градієнта потребує принаймні N + 1 оцінку функцій. Для наближень 2-х похідних (вони знаходяться у матриці Гессе) число оцінок функцій буде порядку N^2. Наприклад, метод Ньютона вимагає похідних 2-го порядку, тобто для кожної ітерації кількість викликів функцій має порядок N^2, проте для більш простого чистого градієнтного оптимізатора потрібно лише N. Однак оптимізатори градієнтів потребують зазвичай більше ітерацій, ніж алгоритм Ньютона. Яка з них буде найкращою за кількістю викликів функцій залежить від конкретної задачі. Отже маємо такі методи.

- Методи, які оцінюють гессіан (методи 2-го порядку):
  - метод Ньютона для оптимізації – відомий ітераційний чисельний метод для знаходження нуля функції;
  - послідовне квадратичне програмування: метод на основі Ньютона для проблем малого та середнього масштабу, деякі версії якого можуть впоратись з багатовимірними проблемами;
  - методи внутрішньої точки: клас методів для умовної оптимізації а також для задач лінійної та нелінійної опуклої оптимізації (англ. convex optimization).
- Методи, які оцінюють градієнт, або апроксимують значення градієнту (методи 1-го порядку):
  - методи координатного спуску: алгоритми, які оновлюють одну координату за одну ітерацію;
  - метод спряжених градієнтів;
  - градієнтний спуск;
  - субградієнтні методи.

- Методи, які оцінюють тільки значення функцій. До таких методів належать і евристичні алгоритми.

Розглянемо відомі евристики для задачі оптимізації.

1. Генетичний алгоритм (ГА). Це еволюційний алгоритм пошуку, основним принципом якого є використання механізмів, що є аналогічними до біологічної еволюції, для розв'язання задач оптимізації. Для цього виконується послідовний підбір, комбінування і варіації шуканих параметрів. Особливістю генетичного алгоритму є використання так званого оператора «схрещення», який виконує операцію рекомбінації (тобто складає нове рішення з існуючих), роль якої аналогічна ролі схрещення при розмноженні в живій природі.

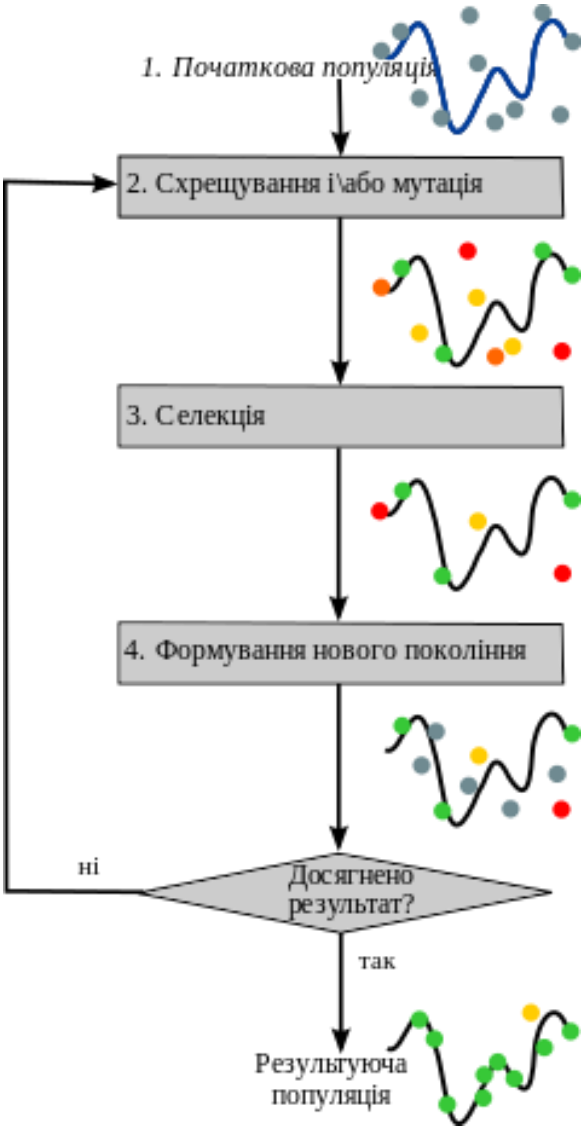

Рисунок 1.1 - Схема генетичного алгоритму

2. Диференційна еволюція (ДЕ). ДЕ також відноситься до класу еволюційних алгоритмів, який можна описати наступним чином. Спочатку генерується деяка множина розв'язків, так зване покоління. Під розв'язками розуміються точки n-вимірного простору, в якому визначена цільова функція, яку потрібно мінімізувати. На кожній ітерації алгоритм формує нове покоління рішень, випадковим чином комбінуючи розв'язки з попереднього покоління. Число рішень в кожному поколінні одне й те саме і є одним з параметрів алгоритму. Нове покоління генерується в такий спосіб: для кожного розв'язку X зі старого покоління вибираються декілька різних випадкових рішень серед старого покоління, за винятком самого розв'язку X i генерується так зване мутантне рішення, над яким виконується операція «схрещування», яка полягає в тому, що деякі його координати заміщаються відповідними координатами з початкового розв'зку X (кожна координата заміщається з деякою ймовірністю, яка також є ще одним з параметрів цього методу). Отриманий таким чином після схрещування розв'язок називають пробним. Якщо він виявляється кращим за X (тобто значення цільової функції стало меншим), то в новому поколінні X замінюється на пробний розв'язок, а в іншому разі — залишається.

3. Метод рою часток (МРЧ). Це метод чисельної оптимізації, який оптимізує функцію, підтримуючи популяцію можливих рішень, називаних частками, і переміщаючи ці частки в просторі розв'язків згідно із відповідною формулою. Переміщення підпорядковуються наступному принципу: знаходиться найкраще в цьому просторі положення, яке постійно змінюється при знаходженні частками вигідніших положень.

4. Алгоритм бджолиної колонії (АБК). Щоб зрозуміти принципи роботи бджолиного алгоритму, розглянемо поведінку реальної бджолиної сім'ї (колонії), яка зустрічається в природі. Всі бджоли колонії діють індивідуально

відповідно до наступного принципу: швидкість польоту бджоли збільшується в напрямку найкращої персональної і найкращої спільної позиції, постійно перевіряючи значення поточної позиції. Пошук відбувається в деякому N-мірному просторі, який є областю рішень для задачі, що оптимізується, де кожний набір координат представляє розв’язок. За аналогією з реальною бджолиною колонією функцією придатності буде щільність квітів, тобто чим більша щільність, тим краща позиція. Функція придатності слугує засобом зв'язку між фізичною проблемою і алгоритмом оптимізації. Кожна бджола пам'ятає позицію, де вона сама виявила найбільшу кількість квітів, яку будемо називати найкращою персональною позицією (НПП). Це позиція з найбільшим значенням придатності, виявлена бджолою. Кожна бджола має власне НПП, яке визначається шляхом, який вона пролетіла. Також у кожній точці вздовж свого руху бджола порівнює значення придатності поточної позиції зі значенням НПП. Якщо поточна позиція має значення придатності вище, значення персональної найкращої позиції замінюється на значення поточної позиції. Також кожна бджола колонії може дізнаватись про область найбільшої концентрації квітів за допомогою відомого в природі «бджолиного танцю», отже результуючий розв’язок визначається від танців даної бджоли з кожної з бджіл-розвідниць. Якщо одна із бджіл бачить, що джерело квітів, знайдене іншою бджолою значно краще, ніж знайдене нею, вона летить для перевірки цих даних. Якщо це дійсно так, то по прильоту до вулика, вона починає переконувати бджіл збирати нектар в новій області найбільшої концентрації квітів. Така позиція найбільшої придатності має назву найкраща глобальна позиція (НГП). Для всієї колонії існує одна глобальна найкраща позиція, до якої прагне кожна бджола. У кожній точці протягом всього шляху кожна бджола порівнює придатність її поточної позиції з НГП. Якщо будь-яка бджола виявить кращу позицію, відповідно до описаної процедури НГП замінюється поточною позицією.

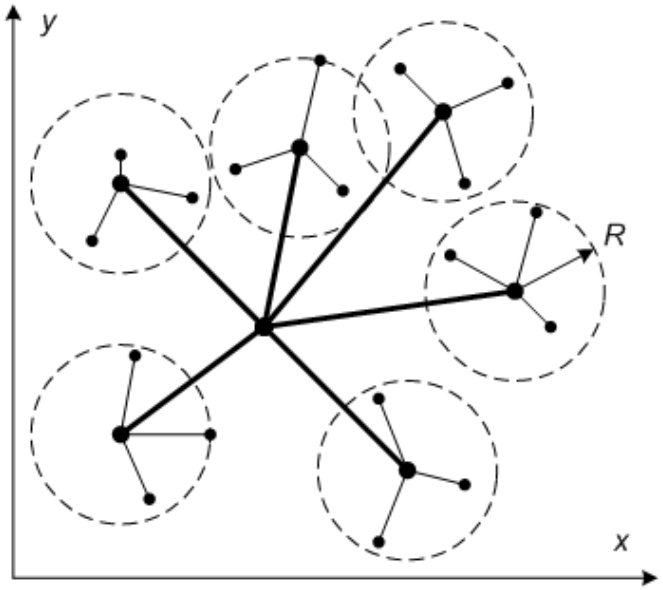

Рисунок 1.2 – Принцип бджолиного алгоритму

5. Алгоритм імітації відпалу (AIB). Це алгоритм, в якому процес пошуку глобального екстремуму імітує фізичний процес відпалу, який полягаю у наступному. У рідкій фазі частинки (атоми та молекули) розташовуються випадковим чином, а в основному стані твердого тіла усі частинки розташовані в добре структуровані ґратки, для яких відповідна їх енергія мінімальна. Ми отримуємо основний стан твердого тіла тільки тоді, коли максимальне значення температури досить високе і охолодження здійснюється досить повільно. В іншому випадку, тіло застигне в деякому метастабільному стані, а не в істинному. Отже, даний алгоритм ґрунтується на імітації фізичного процесу, який відбувається при кристалізації речовини з рідкого стану в твердий, у тому числі при відпалі металів. Передбачається, що атоми вже вишикувалися в кристалічну решітку, але ще допустимі переходи окремих атомів з однієї комірки в іншу. Такий процес протікає при поступовому зниженні температури. Перехід частинки з однієї комірки в іншу відбувається з деякою ймовірністю, причому вона зменшується з пониженням температури. Стійка кристалічна решітка відповідає мінімуму енергії атомів,

тому атом або переходить в стан з меншим рівнем енергії, або залишається на місці.

6. Табу-пошук (ТП). ТП є мета-евристичним алгоритмом, принцип якого полягає у наступному. Ведеться локальний пошук, і щоб запобігти його від попадання в пастку в передчасних локальних ектремумах, забороняються ті переміщення, які змушують повертатися до попередніх розв'язків і циклічної роботи. ТП починається з випадкового початкового рішення. На кожній ітерації генерується набор рішень, і найкращі з цього набору вибираються як нові рішення. Певні атрибути попередніх рішень зберігаються в так званому табу-списку, який оновлюється в кінці кожної ітерації. Вибір найкращого рішення відбувається таким чином, що він не приймає жодного з заборонених атрибутів, які зберігаються в табу-списку. Найкраще допустиме рішення на даний момент оновлюється, якщо нове поточне рішення краще і допустиме. Процедура триває до тих пір, поки не виконається будь-який з двох критеріїв зупинки алгоритму, якими є максимальне число виконуваних ітерацій і максимальне число ітерацій, під час яких чинне рішення не поліпшується.

7. Мурашиний алгоритм (МАО). Принцип МАО полягає в моделюванні реальної поведінки мурах в природі. Початково мурахи блукають в просторі довільним чином, і по знаходженні їжі повертаються до своєї колонії, залишаючи по собі феромонний слід. Після цього якщо інші мурахи знаходять такий шлях, вони схильні до припинення своїх блукань, тобто вони починають слідувати позначеним шляхом, посилюючи його під час повернення у разі знайдення їжі. Проте, з часом, феромонові сліди випаровуються, одже привабливість шляхів зменшується. Тобто очевидно, чим більше часу потрібно мурасі, щоб подолати дорогу, тим більше часу мають феромонові сліди, щоб випаруватись. Натомість, коротка дорога проходиться швидше (відповідно і частіше), отже щільність феромонів стає

більшою на короткому шляху. Випаровування феромонів також надає перевагу уникнення локально найкращих шляхів. Якби випаровування не відбувалось взагалі, шляхи обрані першим мурахою ставали б вкрай привабливими для наступних, тобто розвідка можливих шляхів була б обмежена. Таким чином, коли мураха знаходить вдалий (тобто коротший з наявних) шлях з колонії до джерела їжі, інші мурахи більш ймовірно слідуватимуть йому, і такий позитивний зворотний зв'язок в підсумку призведе до обрання цього кращого шляху всіма мурахами. Отже, ідея мурашиного алгоритму полягає в наслідуванні поведінки колонії мурах, що прогулюються графом, який представляє проблему для розв'язання.

Також можна виділити деякі варіації мурашиного алгоритму:

- Елітна система мурашок (англ. Elitist ant system). Ідея цієї системи полягає в тому, що глобальне найкраще рішення поширює феромони на кожну ітерацію для всіх інших мурах.
- Максимальна система (англ. Max-min ant system, MMAS). Головна відмінність цієї системи в тому, що додаються максимальні та мінімальні кількості феромонів [Tmax, Tmin].
- Рангова система мурашок (англ. Rank-based ant system, ASrank). Всі рішення ранжируються відповідно до їх довжини. Кількість випареного феромону потім зважується для кожної мурахи, так що мурахи з більш короткими шляхами відкладають більше феромонів, ніж особи з більш довгими шляхами.
- Неперервна ортогональна колонія мурашок (англ. Continuous orthogonal ant colony, COAC). Механізм даного алгоритму полягає в тому, щоб дозволити мурашкам шукати рішення спільно і ефективно використовуючи метод ортогонального проектування

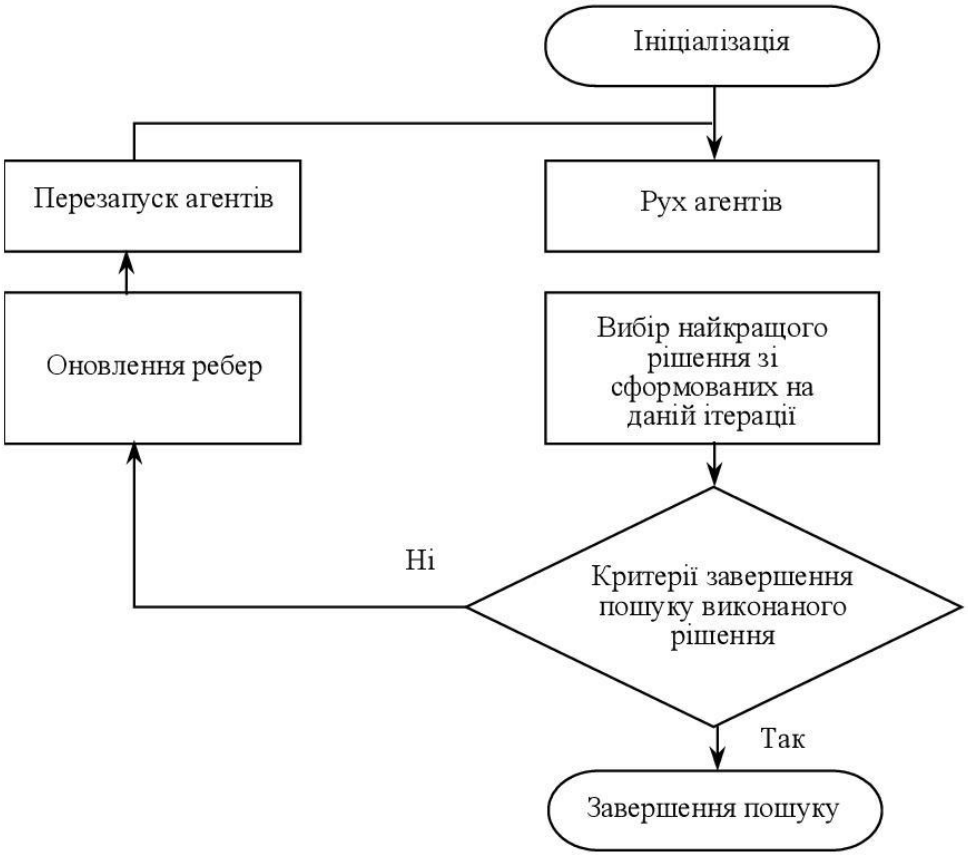

Рисунок 1.3 - Принцип колективного інтелекту, який використовується для евристичних алгоритмів оптимізації.

1.3. Обґрунтування теми дипломного роботи

Евристичні алгоритми оптимізації мають широке застосування для розв’язання прикладних завдань, наприклад, задача квадратичного призначення, задача комівояжера, задача календарного планування, завдання маршрутизації транспорту, різних мереж (GPRS, телефонні, комп'ютерні тощо), розподіл ресурсів та робіт, оптимізація форми антен, наприклад, RFID-тегів. Ці задачі виникають у бізнесі, інженерії, виробництві та багатьох інших областях.

Мурашиний алгоритм є достатньо молодим, адже перші дослідження його почались лише в 90-х роках XX століття. Головним ідеологом алгоритму є Марко Доріго, який запропонував систему мурах в своїй докторській

дисертації в 1992 році. Попередні висновки показують , що метод мурашиних колоній може давати результати, навіть кращі, ніж при використанні генетичних алгоритмів і нейронних мереж, отже саме цей алгоритм є найперспективнішим для нових розробок.

Наприклад, запропонований К. Соча і М. Доріго в 2006 році мурашиний алгоритм оптимізації в неперервному просторі (МАНП), показав досить непогані результати, багато в чому кращі ніж у інших алгоритмів. В таблиці 1.1 продемонстроване порівняння деяких відомих реалізацій евристик з цим алгоритмом. В даній таблиці генетичний алгоритм (ГА) представлений в реалізації Челуа і Сіарі, 2000 р., табу-пошук (ТП) – Челуа і Сіарі, 1999 р., імітація відпалу (АІВ) – Сіарі, 1997 р., диференційна еволюція (ДЕ) – Сторн і Прайс, 1995р.

Тестові функції	МАНП	ГА	ТП	АІВ	ДЕ
Branin RCOS	3.5	2.5	**1.0** (245)	–	–
B_2	1.3	**1.0** (430)	–	–	–
Easom	**1.0** [98%] (772)	1.9	–	–	–
Goldstein and Price	1.7	1.8	**1.0** (231)	3.4	–
Rosenbrock (R_2)	1.7	2.0	**1.0** (480)	1.7	1.3
Zakharov (Z_2)	1.5	3.2	**1.0** (195)	81	–
De Jong	1.0	1.9	–	–	**1.0** (392)
Hartmann ($H_{3,4}$)	**1.0** (342)	1.7	1.6	2.0	–
Shekel ($S_{4,5}$)	1.3 [57%]	**1.0** [76%] (610)	1.4 [75%]	1.9 [54%]	–
Shekel ($S_{4,7}$)	1.1 [79%]	**1.0** [83%] (680)	1.3 [80%]	1.8 [54%]	–
Shekel ($S_{4,10}$)	1.1 [81%]	**1.0** [83%] (650)	1.4 [80%]	1.8 [50%]	–
Rosenbrock (R_5)	1.2 [97%]	1.9	**1.0** (2142)	2.5	–
Zakharov (Z_5)	**1.0** (727)	1.9	3.1	96	–
Hartmann ($H_{6,4}$)	**1.0** (722)	1.3	2.1	3.7	–
Griewangk (Gr_{10})	**1.0** [61%] (1390)	–	–	–	9.2

Таблиця 1.1 - Результати, отримані за допомогою мурашиного алгоритму, порівняно з результатами, отриманими іншими евристиками, адаптованими до безперервних областей

В таблиці 1.1 зазначена відносна середня кількість обчислювань функцій. Фактичне середнє число обчислювань функцій наведено в дужках тільки для найбільш ефективного алгоритму з даної проблеми. Числа в квадратних дужках вказують на відсоток успішних прогонів. Коли відсоток не

вказано - всі пробіги були успішними. Зауважимо, що для деяких алгоритмів результатів на деяких тестових функціях не було доступні.

Як видно з цієї таблиці, мурашиний алгоритм для більшості тестових функцій показує найкращі та наближені до кращих результати.

В даній дипломній роботі для рішення задачі оптимізації вводяться так звані агрегаційні феромони, які також спостерігаються в природі, і пропонується мурашиний алгоритм з метафорою агрегації феромонів.

2. ОПИС МУРАШИНОГО АЛГОРИТМУ ТА СИСТЕМИ АГРЕГАЦІЇ ФЕРОМОНІВ

2.1. Мурашиний алгоритм оптимізації

Як було зазначено вище, концепція мурашиного алгоритму була запропонована для рішення задачі комівояжера. Для того, щоб зрозуміти принцип роботи алгоритму, розглянемо рисунок 2.1.

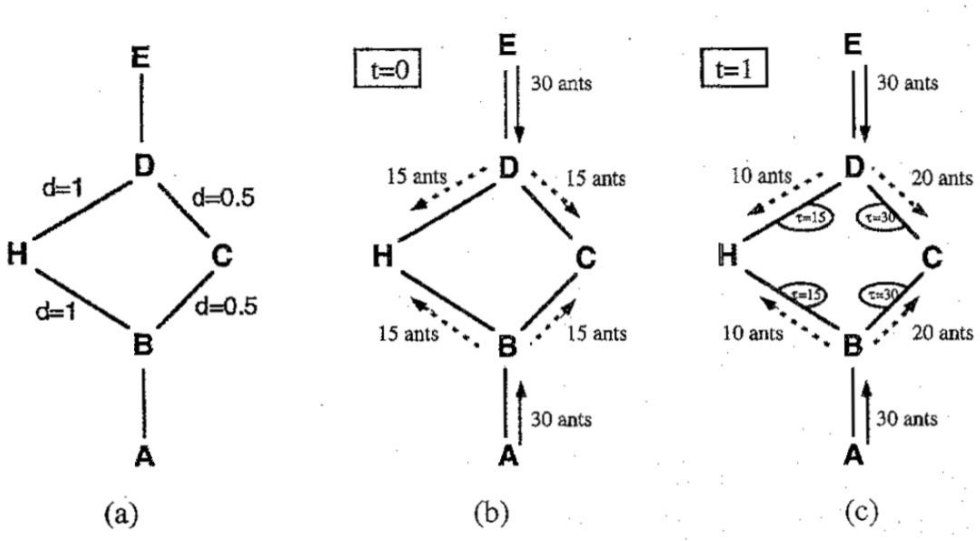

Рисунок 2.1 - Концепція МАО.

Припустимо, що відстані DH, BH дорівнюють 1, а відстані BC, DC – 0,5. Тепер розглянемо, що відбувається кожний дискретний проміжок часу: t = 0, 1, 2… Отже, 30 нових мурах вирушають від B до A, а також від E до D. Нехай мурахи ходять зі швидкістю 1 на одиницю часу, і під час ходьби випаровуються феромони з інтенсивністю 1.

У момент часу t = 0 на ребрах графа немає жодного сліду, тому 30 мурашок в B і D вибирають чи слід повернути праворуч або ліворуч з однаковою ймовірністю. Тому в середньому 15 мурах з кожного вузла буде йти у напрямку H і 15 до C.

При t = 1 30 нових мурах, які приходять від A до B, знаходять феромоний слід інтенсивності 15 на шляху до вершини H, закладений 15 мурахами, що пройшли раніше цим шляхом, та слід інтенсивності 30 на відрізку BC, отриманий як сума сліду, закладеного 15 мурахами, що пішли з B та 15 мурахами, що досягли B через CD, оскільки відстань на цих відрізках = 0,5. Ймовірність вибору шляху змінюється до 30/15 = 2/1 між варіантами BC і BH відповідно, отже, 20 мурах оберуть шлях до C, а 10 - до H. Те ж саме стосується і нових 30 мурах в D, що походять від E.

Цей процес продовжується до тих пір, поки всі мурахи не прийдуть до вибору найкоротшого шляху. Ідея полягає в тому, що якщо в певній точці мураха має вибрати між різними шляхами, ті, які більше обирались попередніми мурашками, вибираються з більшою ймовірністю.

Тепер розглянемо, як працює МАО для задачі комівояжера.

З урахуванням набору n міст проблема комівояжера – це задача пошуку мінімальної довжини замкненого туру, який проходить через кожне місто один раз. Задача комівояжера задається графом (N, E), де N – міста, і E – сукупність доріг між містами.

Нехай $b_i(t)$ (i = 1, … , n) – число мурашок у місті в час t і нехай $m = \sum_{i=1}^{n} b_i(t)$ – загальна кількість мурашок. Кожен мураха є простим агентом з наступними характеристиками:

- він вибирає місто, щоб перейти з вірогідністю, яка є функцією відстані від міста та кількості сліду на шляху між містами;
- він має здійснювати легальні шляхи, адже переходити у вже відвідані міста заборонені;
- коли він завершує тур, мураха залишає феромоний слід на кожній вершині.

Нехай $\tau_{ij}(t)$ – інтенсивність сліду феромонів на ребрі (i, j) в момент часу t. Кожен мураха під час t вибирає наступне місто, де він буде в час t +1. На цьому етапі щільність феромонів оновлюється за такою формулою:

$$\tau_{ij}(t+n) = \rho \cdot \tau_{ij}(t) + \Delta\tau_{ij} \qquad (1)$$

де ρ - коефіцієнт такий, що (1- ρ) являє собою кількість випаровування феромонів між часом t і t+n,

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau^{k}{}_{ij} \qquad (2)$$

де $\Delta\tau^{k}{}_{ij}$ – кількість на одиницю довжини феромону, закладені на вершину (i, j) k-тою мурахою між часом t і t+n, тобто:

$$\Delta\tau^{k}{}_{ij} = \begin{cases} \frac{Q}{L_k} & \text{якщо } k-\text{та мураха на вершині (i, j)} \\ 0 & \text{в іншому випадку} \end{cases} \qquad (3)$$

де Q константа і L_k – це довжина шляху для k-тої мурахи.

Коефіцієнт ρ повинен бути встановлений на значення менше 1, щоб уникнути необмеженого накопичування сліду.

Для того, щоб задовольнити обмеження, що мураха відвідує всі різні міста, кожна мураха пов’язується з структурою даних, яку називають табу-лист, куди записуються міста що були вже відвідані і забороняється відвідати їх знову до завершення n ітерацій. Після завершення туру, табу-лист використовується для обчислення поточного рішення мурахи (тобто відстань цього шляху). Тоді табу-лист вивільняється і мураха знову вільний на вибір. Визначимо $tabu_k$ як динамічно зростаючий вектор, який містить табу-лист для k-тої мурахи, тоді $tabu_k(s)$ – це s-тий елемент списку.

Визначимо величину видимості як $\eta_{ij} = 1/d_{ij}$, яка не змінюється під час роботи алгоритму , на відмінну від функції випаровування феромонів.

Також визначимо ймовірність переходу від міста і до j для k-тої мурахи, як:

$$p^k{}_{ij}(t) = \begin{cases} \dfrac{\tau^\alpha{}_{ij}(t) \cdot \eta_{ij}{}^\beta}{\sum_{k=1}^{n} \tau^\alpha{}_{ij}(t) \cdot \eta_{ij}{}^\beta} & \text{якщо } j \text{ дозволене місто} \\ 0 \text{ в іншому випадку} \end{cases} \quad (4)$$

де α та β – параметри, що контролюють відносну важливість інтенсивності сліду і видимості. Отже, ймовірність переходу є компромісом між видимістю (яка говорить, що найближчі міста повинні вибиратися з великою ймовірністю) і щільністю феромонного сліду в момент часу t.

Формально МАО для задачі комівояжера це:

1. Ініціалізація:

   t:=0 (t – лічильник часу);

   NC:=0 (NC – лічильник ітерацій циклу);

   для кожного міста (i, j) визначаємо початкове значення $\tau_{ij}(t) = c$

   для інтенсивності сліду і $\Delta\tau_{ij} = 0$;

   розташуємо m мурашок в n вершинах;

2. s:=1 (s – це індекс в табу-листі);

   for k:=1 to m do:

   поміщаєм стартове місто k-тої мурахи в $tabu_k(s)$;

3. repeat until табу-лист заповнений (цей крок повторюється (n-1) разів);

   s:=s+1;

   for k:=1 to m do:

   обираєм місто j для руху з ймовірністю $p^k{}_{ij}(t)$ (в час t k-та мураха знаходиться в $i=tabu_k(s-1)$);

   k-та мураха переходить в місто j;

   вставляєм місто j в $tabu_k(s)$;

4. for k:=1 to m do:

k-та мураха переходить з $tabu_k(n)$ в $tabu_k(1)$;

обчислюємо L_k для k-тої мурахи;

оновлюємо знайдений найкоротший шлях;

для кожної вершини (i, j):

for k:=1 to m do:

$$\Delta\tau^k{}_{ij} = \begin{cases} \frac{Q}{L_k} & \text{якщо } (i,j) \in tabu_k \\ 0 & \text{в іншому випадку} \end{cases}$$

$$\Delta\tau_{ij} := \Delta\tau_{ij} + \Delta\tau^k{}_{ij}$$

5. для кожної вершини (i, j) розраховуємо $\tau_{ij}(t+n)$ відповідно для рівняння $\tau_{ij}(t+n) = \rho \cdot \tau_{ij}(t) + \Delta\tau_{ij}$;

   t:=t+n;

   NC:=NC+1;

   для кожної вершини (i, j) встановлюємо $\Delta\tau_{ij} := 0$;

6. якщо ($NC < NC_{max}$) і нема стагнації поведінки то:

   звільнити всі табу-листи;

   до кроку 2;

   інакше:

   показати найкращій шлях;

   стоп;

Складність алгоритму буде дорівнювати $O(NC \cdot n^2 \cdot m)$, якщо алгоритм зупиняється на NC-тій ітерації.

2.2. Система агрегації феромонів

Існує деяка різниця між МАО і САФ яка полягає в тому, як феромони функціонують у просторі пошуку. Щільність феромонів (інтенсивність сліду) в МАО визначається як слід на шляху між вузлами заданої послідовності

вузлів (міст для задачі комівояжера). У САФ щільність феромонів агрегації визначається функцією щільності в просторі пошуку X в R^n . У реальному світі феромони агрегації використовуються різними видами для спілкування з членами своєї спільноти для обміну інформацією про місцезнаходження їжі, безпечного притулку, потенційних партнерів або ворогів.

Як і в мурашиному алгоритмі оптимізації, САФ приймає модель циклу. У кожному циклі САФ, запозичений із природного моделі, m мурахи залучають агрегацію феромонів до вибору позицій в просторі пошуку X. Вони більше рухаються до позицій, де щільність феромонів вище, і менше до тих позицій, де щільність нижче. Таким чином, в нашій системі ми розглядаємо окремих осіб для вибору своїх позицій в залежності від агрегатної щільності феромонів, а точніше, з імовірністю, пропорційною функції щільності агрегації в просторі пошуку X наступним чином.

Нехай x є змінною у просторі пошуку X , тобто $x \in X$, $\tau(t, x)$ - функція щільності агрегація феромонів. У початковому стані САФ ($t = 0$) феромони агрегації розподіляється рівномірно, тобто $\tau(0, x) = c$, де, c - константа. Функція щільності ймовірності (ЩЙ) для агрегаційних феромонів визначається як:

$$p_\tau(t, x) = \frac{\tau(t, x)}{\int_X \tau(t, x)dx} \qquad (5)$$

Залежно від проблеми, яку ми вирішуємо функція $f(x)$ призначається для $x \in X$. Кожен з m мурах випускають агрегаційні феромони навколо свого положення x в X залежно від фітнес-функції $f(x)$. Можуть виникнути питання щодо того, який підхід більш схожий на той, який використовується в природному світі, для використання абсолютного значення $f(x)$, відносного значення або рейтингової системи. Однак у нашій системі ми

використовуємо ранг r ($r = 1, 2,\ldots, m$) кожного індивідуума, щоб зробити різницю придатності між особами більш помітною. Мураха, яка обрала найкраще рішення має значення m, а найгірше - має значення 1. Той, хто має ранг r випаровує феромони навколо $x_{t,r}$ з щільністю представленою функцією $\Delta\tau'(t, r, x_{t,r}, x)$, де $x_{t,r}$ це позиція мурахи з рангом r. Тобто щільність агрегаційних феромонів, що були випущені в момент часу t m індивідуумами має такий вигляд:

$$\Delta\tau(t, x) = \sum_{r=1}^{m} \Delta\tau'(t, r, x_{t,r}, x) \qquad (6)$$

Отже, тут ми припускаємо, що сумарний агрегаційний обсяг феромонів, що випускається m мурахами в кожний момент часу t дорівнює C, тобто:

$$\int_X \Delta\tau(t, x)dx = \mathrm{C} \qquad (7)$$

Це припущення важливе для методу вибору кращих рішень. Коли всі m мурах завершують свій вибір, то загальна агрегаційна щільність феромонів в X оновлюється за такою формулою:

$$\Delta\tau(t + 1, x) = \rho \cdot \tau(t, x) + \Delta\tau(t, x) \qquad (8)$$

Після виконання оновлення феромонів індивідууми скидаються і починається наступна ітерація САФ. Цей процес повторюється. Таким чином, щільність феромонів у перспективних точках у просторі збільшується по мірі ітерації циклу САФ, і, як очікується, система агрегації феромонів сходиться до кращого рішення.

Ще раз розглянемо формулу $\Delta\tau'(t, r, x_{t,r}, x)$ рівняння 6. Будемо вважати, що $\Delta\tau'(t, r, x_{t,r}, x)$ є в центрі навколо положення $x_{t,r}$ (положення $x_{t,r}$ має найвищу щільність), мураха з більш високим рангом виділяє більше феромона, ніж індивід з нижчим рангом. Крім того, ми вводимо кооперативне явище , в якому $\Delta\tau'(t, r, x_{t,r}, x)$ впливає на розподіл інших індивідуумів і витягується в напрямку розподілу. У цьому дослідженні ми використовуємо наступну функціональну форму Гауса для $\Delta\tau'(t, r, x_{t,r}, x)$ (r = 1, 2,…, m) для рівняння 6:

$$\Delta\tau'(t, r, x_{t,r}, x) = \frac{C}{\sum_{k=1}^{m} k^{\alpha}} r^{\alpha} N(x_{t,r}, \beta^2 S_t) \qquad (9)$$

де $\alpha > 0$ – параметр, який відображає відносну важливість рангу, S_t-коваріаційна матриця, яка оцінюється від розподілу m індивідуумів у просторі пошуку X, $\beta > 0$ – параметр, що контролює ширину росподілу феромонів, $N(x_{t,r}, \beta^2 S_t)$ – є багатомірним нормальним росподілом.

Функція, визначена формулою 9 задовольняє всім трьом вищевказаним умовам. Вона має найвище значення в точці $x_{t,r}$, мурахи більш високого рангу мають більші значення функції і коваріаційної матриці $\beta^2 S_{tm}$, , яка відображає розподіл осіб. При великих значеннях α, відношення загальної кількості феромону виробленого мурахами більш високого рангу збільшується. При більших значеннях β феромон агрегації поширюється більш широко в просторі пошуку X , а при менших значеннях менш широко,

агрегації менше. Отже, $\Delta\tau(t,x)$, щільність феромонів, випущених m мурахами в час t вираховується за допомогою рівняння 3.

Розглянемо, як формується вибірка кращих рішень в системі агрегації феромонів. Як було зазначено вище, ми відбираємо осіб з ймовірністю, пропорційною щільності агрегації $\tau(t+1,x)$. Для реалізації пропорційної вибірки нам потрібно отримати функцію щільності ймовірності (ЩЙ) $p_\tau(t+1,x)$ з $\tau(t+1,x)$. $\tau(t+1,x)$ з рівняння 8 можна представити як:

$$p_\tau(t+1,x) = \rho^{t+1}\tau(0,x) + \sum_{h=0}^{m}\rho^h\Delta\tau(t-h,x). \qquad (10)$$

З рівнянь 5, 7 і 10 маємо:

$$p_\tau(t+1,x) = \frac{\rho^{t+1}}{\sum_{k=0}^{t+1}\rho^k}\cdot\frac{\tau(0,x)}{C} + \sum_{h=0}^{t}\frac{\rho^h}{\sum_{k=0}^{t+1}\rho^k}\cdot\frac{\Delta\tau(t-h,x)}{C} \quad (11)$$

В загальному випадку, якщо функцію щільності ймовірності (ЩЙ) *P(x)* модна розкласти на підфункції $F_K(x)P(x) = p_1f_1(x) + p_2f_2(x) + \cdots + p_sf_s(x)$, то ми можемо відібрати $f(x)$ наступним чином: вибравши $f_s(x)$ (s=1, 2, …, S) з ймовірностями $(p_1, p_2, \ldots, p_s)$, потім підбираємо $f_s(x)$, де $(p_1, p_2, \ldots, p_s)$ – розподіл ймовірностей. Використовуючи цей метод, можна відібрати $p_\tau(t+1,x)$ наступним чином. В рівнянні 8, $\tau(0,x)/C$ і $\Delta\tau(t-h,x)/C$ можуть бути представлені як в рівнянні 7. Тобто, спочатку ми вибираєм $\tau(0,x)/C$ чи $\Delta\tau(t-h,x)/C$ (h=0,1,…,t) з ймовірністю $\rho^h/\sum_{k=0}^{t+1}\rho^k$ (h=0,1,…,t, t+1). Назвімо це циклічною вибіркою. Отже, з рівняння 9 маємо:

$$\frac{\Delta\tau(t-h,x)}{C} = \sum_{r=1}^{m}\frac{r^\alpha}{L}N\left(x_{t-h,r}, \beta^2 S_{t-h}\right) \qquad (12)$$

де $L = \sum_{k=1}^{m}k^\alpha$. Як і в циклічній вибірці, ми спочатку вибираємо ранг r відповідно до ймовірності r^α/L (r=1, 2, …, m). Це рангова вибірка. Після неї, підбираємо $N\left(x_{t-h,r}, \beta^2 S_{t-h}\right)$ використовуючи декомпозицію Холеського. Для реалізації цього, нам потребується багато пам'яті для збереження $x_{t-h,r}$, значень вектора та кавариційної матриці $\beta^2 S_{t-h}$, коли значення tстануть

великими. Отже, при $\rho^h \to 0$, для великих h з $0 < \rho < 1$, ми можемо прийняти ліміт ітерацій циклу = H. Тобто, рівняння 10 для $t \geq H$ набуває виду:

$$p_\tau(t+1,x) = \sum_{h=0}^{H-1} \frac{\rho^h}{\sum_{l=0}^{H-1} \rho^l} \cdot \frac{\Delta\tau(t-h,x)}{C}$$

3. РОЗРОБКА ПРАКТИЧНОЇ РЕАЛІЗАЦІЇ МУРАШИНОГО АЛГОРИТМУ З МЕТАФОРОЮ АГРЕГАЦІЇ ФЕРОМОНІВ ДЛЯ ГЛОБАЛЬНОЇ ОПТИМІЗАЦІЇ

3.1. Методика реалізації алгоритму для задачі оптимізації в неперервному просторі

Оптимізація в неперервному просторі – це задача знаходження глобального екстремуму деякої цільової функції. Отже, розглянувши все вище написане, можна приступати до практичної реалізації мурашиного алгоритму з метафорою агрегації феромонів для задачі оптимізації в неперервному простору.

Для нашої розробки визначимо, що шлях, який проходять мурахи до цілі, представляє собою рішення, тобто значення цільової функції. Тобто для деякої n-мірної функції $f(x)$ рішенням є її значення $f(x_1, x_2, \dots, x_n)$, а змінні $x_1, x_2, \dots, x_n$ – це мурахи, які визначають саме це рішення.

Нехай ми маємо k наборів змінних (за аналогією, можна сказати, що це k спільнот мурах). Спочатку вони ініціалізуються випадковими числами в просторі пошуку X і обчислюється значення цільової функції для кожного набору змінних. Розв'язки впорядковуються відповідно до їхньої якості, тобто, для завдання мінімізації:

$$f(x_{1,1}, x_{1,2}, \dots, x_{1,n}) \leq f(x_{2,1}, x_{2,2}, \dots, x_{2,n}) \leq \cdots \leq f(x_{k,1}, x_{k,2}, \dots, x_{k,n})$$

Кожний розв'язок має відповідний ранг l і вагу ω пропорційну рангу, тобто кожен розв'язок оцінюється (обчислюється фітнес-функція). Вага є параметром, який визначає ймовірність відбору кожного рішення рангу l наступним чином:

$$p_l = \frac{\omega_l}{\sum_{r=1}^{k} \omega_r} \qquad (1)$$

Вагу ω_l розв'язку з рангом l будемо розраховувати по наступній формулі:

$$\omega_l = \frac{1}{qk\sqrt{2\pi}} e^{-\frac{(l-1)^2}{2q^2k^2}} \qquad (2)$$

яка по суті визначає вагу як значення нормального розподілу функції щільності ймовірностей (функція Гауса) з аргументом l, середнім 1, і стандартним відхиленням qk, де q – параметр алгоритму. Коли q невелике, найкращі розв'язки відбираються з найбільшою ймовірністю, а коли воно велике, ймовірність стає більш однорідною (аналогічно до параметру видимості $\propto$ в МАО).

Формування нових рішень (нових шляхів) відбувається наступним чином. Для кожного i-того виміру $(1, 2, \dots, n)$ відбирається розв'язок l з поміж тих, які вже в нас є, з ймовірністю яка була визначена в формулі 1, потім визначаємо новий аргумент i нового розв'язку як гаусова випадкова величина (тобто та, яка вибирається згідно нормального розподілу щільності ймовірностей) з математичним очікуванням $\mu_{l,i} = x_{l,i}$ і середньоквадратичне відхилення $\sigma_{l,i}$, яке вираховується за формулою:

$$\sigma_{l,i} = \varepsilon \sum_{e=1}^{k} \frac{|x_{e,i} - x_{l,i}|}{k-1}$$

де $\varepsilon > 0$ – це параметр, який однаковий для всіх координат і має той самий ефект, що і коефіцієнт випару феромонів в стандартному мурашиному алгоритмі (або оберненому йому коефіцієнту щільності сліду). Чим більше значення ε, тим нижче швидкість збіжності алгоритму. Цей принцип знаходження нових шляхів аналогічний принципу, який використовується в системі агрегації феромонів. Тобто феромони з більшою ймовірністю концентруються саме біля рішень, які мають вищий ранг.

Відновлення рівня феромонів здійснюється додаванням набору нових згенерованих рішень до початкового набору з наступним видаленням такого ж числа гірших розв’язків. У такий спосіб розмір архіву залишається постійним. Ця процедура гарантує, що в системі зберігаються кращі розв'язки, дозволяючи, тим самим, ефективно направляти мураху в просторі пошуку.

Ці дії повторюються і у підсумку значення цільової функції буде мінімізоване з деякою похибкою.

3.2. Структура розробленої програми

Програма, яка була створена для реалізації мурашиного алгоритму з метафорою агрегацією феромонів, була написана на мові C++, та складається з таких структурних елементів:

- об’єкт AntAlg() – головний об’єкт програми, ініціалізується наступними параметрами:
  1. n_alg – кількість вимірів цільової функції,
  2. k_alg – кількість рішень,
  3. k_del – параметр, який визначає кількість нових розв’язків, що виникають на кожній ітерації,
  4. q_alg – параметр q алгоритму,
  5. xi – параметр ε алгоритму,
  6. max_count – максимальна кількість ітерацій;
- struct component {

  vector <double> args – масив змінних;

  double func – значення цільової функції;

  double weight – вага цього розв’язку;

  double posib – ймовірність вибору цього розв’язку;

  };

- vector <component> T_mas – масив, який представляє собою сукупність усіх розв’язків, що існують на даний момент;
- метод gaussian_random (double mue, double sigma) – відповідно до нормального розподілу щільності ймовірностей (функція Гауса) обирає випадкове число типу double, параметрами цього методу є mue – математичне очікування, sigma - стандартне відхилення;
- метод gauss_func (double mue, double sigma, double x) – повертає розраховану функцію Гауса, де mue – математичне очікування, sigma - стандартне відхилення;
- метод make_T(double hypercube_a, double hypercube_b, int func_numb) – створює початкову конфігурацію алгоритму, тобто заповнює масив T_mas таким чином, що присвоює змінним кожного набору випадкові значення в межах гіперкубу між визначеними параметрами (hypercube_a, hypercube_b), також першочергово обчислює задану параметром func_numb цільову функцію для кожного набору і сортує рішення;
- метод make_weight() – обчислює ваги всіх знайдених розв’язків згідно за формулою;
- метод random(double min, double max) – генерує випадкове значення типу double в інтервалі (min, max);
- метод T_sort() – сортує масив розв’язків згідно значень цільової функції;
- метод make_choice() – виконує відбір рішення для формування нового набору згідно з вище означеним алгоритмом;
- метод make_sigma(double xi, int l, int i) – формує стандартне відхилення для функції gaussian_random(double mue, double sigma) за вище означеною формулою;
- метод run() – запускає алгоритм.

4. АНАЛІЗ РЕЗУЛЬТАТІВ РОБОТИ АЛГОРИТМУ

4.1. Дослідження параметрів алгоритму

Розглянемо цільові функції, які ми будемо використовувати для тестувань алгоритму:

- SPHERE FUNCTION

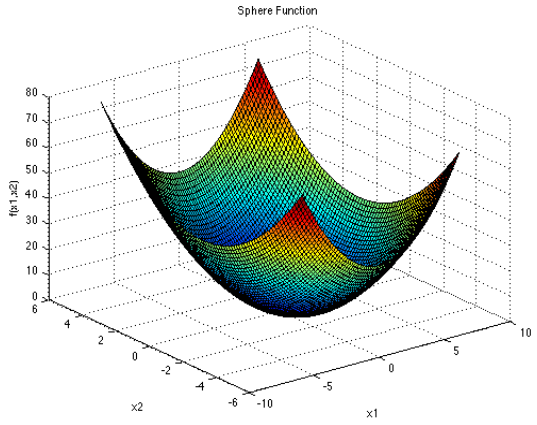

Рис. 4.1. Функція sphere.

Для цієї функції локальний і глобальний мінімум один. Функцію визначаєм у гіперкубі $x_i \in [-100,100]$ для $i = 1,2,...d$,

$$f(x) = \sum_{i=1}^{d} x_i^2 .$$

Глобальний мінімум $f(x^*) = 0$ у точках $x^* = (0,...,0)$.

- ACKLEY FUNCTION

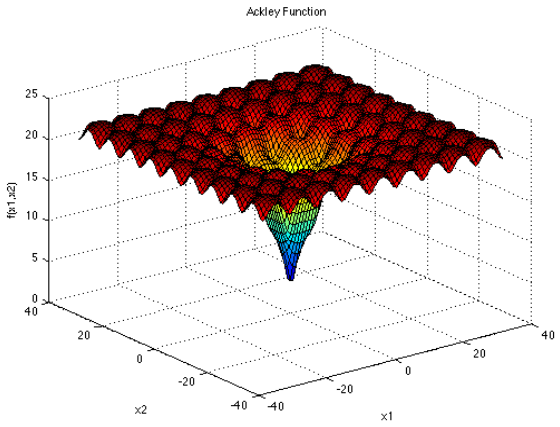

Рис. 4.2. Функція ackley.

Ця функція створює ризик, що алгоритм оптимізації буде захоплений в одному із багатьох локальних мінімумів.

$$f(x) = -ae^{-b\sqrt{\left(\frac{1}{d}\sum_{i=1}^{d} x_i^2\right)}} - e^{\frac{1}{d}\sum_{i=1}^{d}\cos(cx_i)} + a + e$$

Параметри мають такі значення $a = 20, b = 0.2$, $c = 2\pi$

Функція визначена у гіперкубі $x_i \in [-32.768, 32.768]$ для $i = 1,2,...,d$.

Глобальний мінімум $f(x^*) = 0$ в точках $x^* = (0,...,0)$.

- GRIEWANK FUNCTION

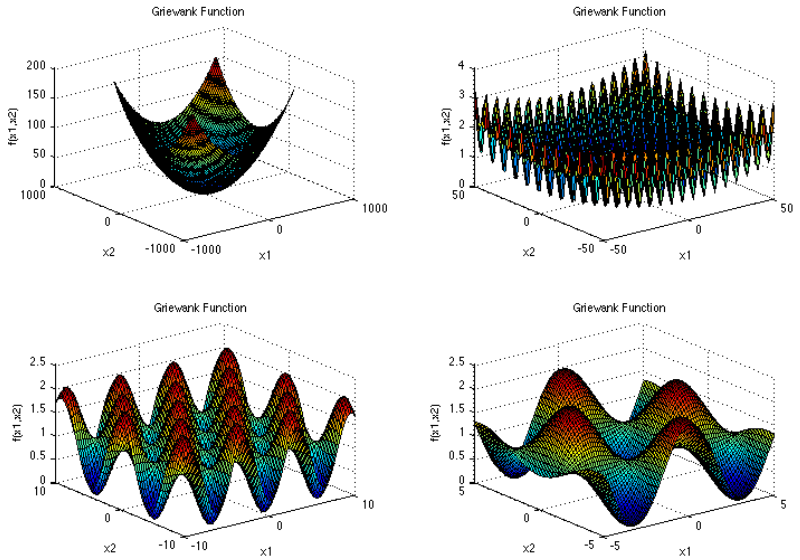

Рис. 4.3 Функція griewank.

Функція griewank має багато поширених локальних мінімумів, які рівномірно розподілені.

$$f(x) = \sum_{i=1}^{d} \frac{x_i^2}{4000} - \prod_{i=1}^{d} cos(\frac{x_i}{\sqrt{i}}) + 1$$

Функція визначена у гіперкубі $x_i \in [-600,600]$. Глобальний мінімум $f(x^*) = 0$ у точках $x^* = (0,...,0)$.

- RASTRIGIN FUNCTION

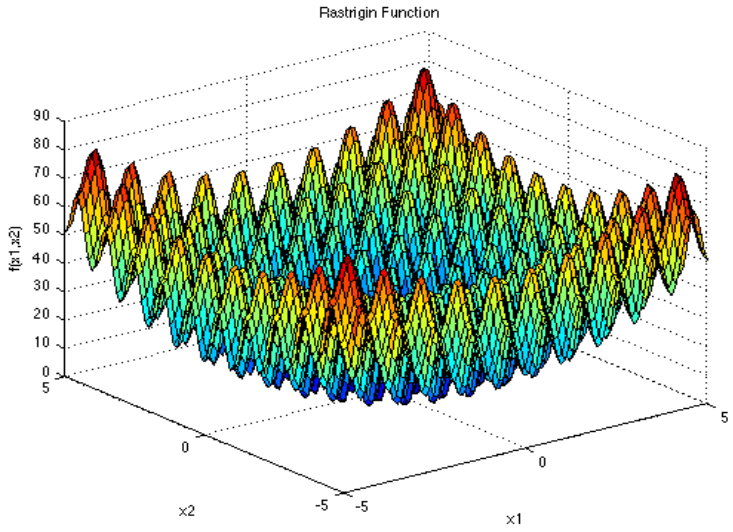

Рис. 4.4 Функція rastrigin.

Функція rastrigin має декілька локальних мінімумів.

$$f(x) = 10d + \sum_{i=1}^{d} (x_i^2 - 10\cos(2\pi x_i)).$$

Функція визначена у гіперкубі $x_i \in [-5.12,5.12]$. Глобальний мінімум $f(x^*) = 0$ в точках $x^* = (0,...,0)$.

- ROSENBROCK FUNCTION

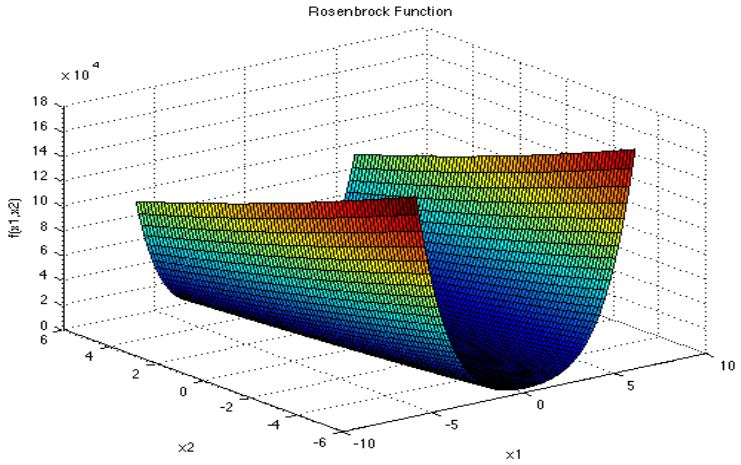

Рис. 4.5 Функція rosenbrock.

У цій функції лежить у вузькій параболічній долині, конвергенція до мінімуму є досить важкою.

$$f(x)=\sum_{i=1}^{d-1}[100(x_{i+1}-x_i^2)^2+(x_i-1)^2].$$

Функція визначена у гіперкубі $x_i \in [-5,10]$ for $i=1,2,...d$. Глобальний мінімум $f(x^*)=0$ при $x^*=(1,...,1)$.

Отже, проведемо такі тестування для дослідження параметрів алгоритму і знайдемо їх оптимальні значення.

Розглянемо, як впливає на роботу алгоритму параметр xi, який ще можна назвати коефіцієнтом випаровування феромонів. Для цього спочатку зробимо загальний огляд параметру xi = (0.1, 0.2, …, 1.5).

k = 10	q = 0.1	xi = 0.2	k_del = 5	max_count = 50	
xi	sphere	ackley	griewank	rastrigin	rosenbrock
0.1	163.156 (122.643)	10.5244 (5.11508)	8.3014 (4.30602)	4.61278 (3.0583)	9.74112 (9.97459)
0.2	129.962 (115.657)	7.87638 (4.1572)	3.69512 (4.83558)	3.24393 (2.81402)	2.51287 (2.29375)
0.3	16.9163 (22.4221)	6.46726 (4.00687)	0.517505 (0.514983)	3.41478 (2.65293)	4.94628 (4.39758)
0.4	6.27971 (9.1536)	1.56091 (2.12168)	0.457885 (0.369669)	2.7945 (1.78735)	0.971931 (1.30481)
0.5	0.415682 (0.72437)	1.73339 (1.38648)	0.354657 (0.298077)	1.99008 (0.994891)	0.667423 (0.649182)
0.6	1.04309 (1.87682)	0.478361 (0.721)	0.293205 (0.234847)	2.78901 (2.3515)	2.46614 (1.98301)
0.7	6.07699e-12 (1.0541e-11)	0.257995 (0.464387)	0.238589 (0.205161)	1.5943 (0.91347)	2.34475 (2.88716)
0.8	3.0046e-06 (5.36673e-06)	7.3975e-06 (7.73725e-06)	0.0608004 (0.0260983)	1.07594 (0.601091)	1.2472 (1.31948)
0.9	6.95897e-10 (1.15566e-09)	1.12246e-05 (9.74165e-06)	0.079566 (0.0275251)	1.27684 (0.473493)	4.16318 (3.4037)
1	4.60245e-08 (5.57346e-08)	6.91535e-05 (6.10117e-05)	0.0705797 (0.0368045)	1.40774 (0.760269)	0.792984 (1.03242)
1.1	1.84262e-07 (1.98314e-07)	0.000462939 (0.000419653)	0.0904439 (0.0414728)	1.43806 (0.602994)	1.30819 (1.67326)
1.2	1.53744e-07 (1.40723e-07)	0.000566558 (0.000492503)	0.0832913 (0.0539112)	1.25378 (0.667297)	0.622725 (0.739796)
1.3	4.44001e-05 (5.06157e-05)	0.00641702 (0.00500811)	0.139215 (0.0546608)	1.272 (0.947478)	8.39669 (9.64412)
1.4	1.13526e-05 (1.45448e-05)	0.0122437 (0.0098573)	0.0970076 (0.0409241)	1.59719 (1.21732)	1.9151 (2.28062)

Таблиця 4.1 – Аналіз параметру xi

Тут (в таблиці 4.1) і надалі в таблицях маємо значення середньої похибки алгоритму при 50 випробуваннях для відповідної функції при max_count ітераціях. В дужках вказане відповідне середнє відхилення. Кількість вимірів для усіх функцій n = 4.

Одним з найважливіших показників алгоритму є кількість обчислювань цільової функції. Позначимо цей показник як N_alg.

В тестуванні, яке продемонстроване в таблиці 4.1, ми маємо max_count=100 ітерацій алгоритму і в кожній ітерації формуються k_del=5 нових рішень, а стартова кількість розв'язків k=10, тобто загальна кількість обчислювань цільової функції складатиме N_alg = 5*100 + 10 = 510.

Отже, як ми бачимо в таблиці 4.1, значення параметру xi, при яких тестові функції (окрім rosenbrock) приймають найменші значення, знаходяться в проміжку (0.9, 1.2), проте знаходження мінімуму функції в цьому тестуванні практично не відбувається. Це зв'язано с тим, що на

кожному кроці роботи, алгоритм має дуже невелику кількість рішень (k = 10) і формує лише k_del = 5 нових розв'язків, у зв'язку з чим max_count = 100 ітерацій недостатньо щоб мінімізувати функції. Для того щоб більше мінімізувати функції, запустимо тестування зі збільшеною варіативністю вибірки k = 200 і k_del = 100.

k = 200	q = 0.1		k_del = 100	max_count = 50	
xi	sphere	ackley	griewank	rastrigin	rosenbrock
0.1	9.32029e-33	0	0.00492987	0	0.0569633
0.2	6.16245e-33	0	0.00246617	0	0.0201665
0.3	2.70714e-30	3.55271e-15	0.00302501	0	5.21649e-05
0.4	3.7333e-27	1.38556e-13	0.00537698	0	8.11904e-05
0.5	9.69445e-25	4.07496e-12	0.00911669	0	0.00276513
0.6	2.16116e-20	1.22206e-10	0.00509109	5.68434e-14	2.4924e-06
0.7	5.18176e-18	1.41291e-09	0.00633429	1.39709e-07	0.000248138
0.8	2.4966e-17	2.97652e-08	0.0107789	0.00138391	0.000231077
0.9	1.47783e-14	4.39088e-07	0.00537889	0.0219025	0.000210728
1	8.8755e-13	8.66737e-08	0.0214785	0.0632468	0.000809022
1.1	3.0974e-11	4.80454e-06	0.00545664	0.0617176	0.00213086
1.2	4.80315e-11	2.19172e-05	0.00930188	0.158622	0.0166149
1.3	2.36957e-10	0.000153851	0.0232244	0.224275	6.50553e-05
1.4	2.58422e-07	0.000386584	0.0256405	0.0827121	0.0180182

Таблиця 4.2 – Аналіз параметру xi

N_alg = 100*50 + 200 = 5200 для цього тестування.

З таблиці 4.2 визначимо найкращі значення параметру xi для наших тестових функцій при k = 200 і k_del = 100. Для функція sphere найкращім значенням параметру xi є 0.2, а також можна визначити таку залежність – чим менше xi, тим швидше відбувається сходження до мінімуму (таку ж залежність можна виділити і для ackley). Проте з попереднього досліду (таблиця 4.1) видно, що така залежність має місце тільки до моменту досягнення оптимального значення xi, після чого зменшуючи цей параметр,

ми не отримаємо пришвидшення алгоритму (це можна побачити і в таблиці 4.2 для функції rosenbrock).

Також визначимо, що зменшення параметру xi мало впливає на швидкість мінімізації функції griewank, на відмінно від інших. Це можна пояснити тим, що ця функція має безліч локальних мінімумів, які дуже близькі до глобального, отже алгоритм може «застрягати» в них. Тобто, оскільки xi має той самий ефект, що і коефіцієнт випару феромонів в стандартному мурашиному алгоритмі, при маленьких значеннях xi щільність феромонів на локальному мінімумі може бути досить великою, що алгоритм буде вважати саме це остаточним рішенням.

Отже, щоб підтвердити наші попередні висновки по впливу параметру xi на швидкість збіжності алгоритму, проведемо наступне тестування з k = 80, k_del = 40 і max_count = 50.

k = 80	q = 0.1	xi = 0.2	k_del = 40	max_count = 50	
xi	sphere	ackley	griewank	rastrigin	rosenbrock
0.2	4.59422e-32	3.55271e-15	0.041915	0	0.0301958
0.4	1.86664e-26	7.10543e-14	0.0135117	0	0.00345584
0.6	2.75671e-20	5.09857e-10	0.00664879	1.57765e-10	0.000504312
0.8	5.48479e-15	1.27122e-07	0.0101966	0.0135244	0.0124487
1	3.78206e-13	6.01599e-06	0.0174552	0.0696411	0.00462004

Таблиця 4.3 – Аналіз параметру xi

N_alg = 40*50 + 80 = 2080 для цього тестування.

Отже, з таблиці 4.3 видно, що швидкість мінімізації griewank практично не залежить від параметру xi на проміжку (0.2; 1).

Побудуємо графік залежності похибки алгоритму від параметру xi (рис. 4.6) для функції sphere. k = 100; q = 0.1; k_del = 50; max_count = 50, а xi змінюється від 0.05 до 2 з кроком 0.05 для цього тестування, тобто N_alg =

50*50 + 100 = 2100. Ось Y в даному графіку, як і в усіх наступних, представлена логарифмічною шкалою.

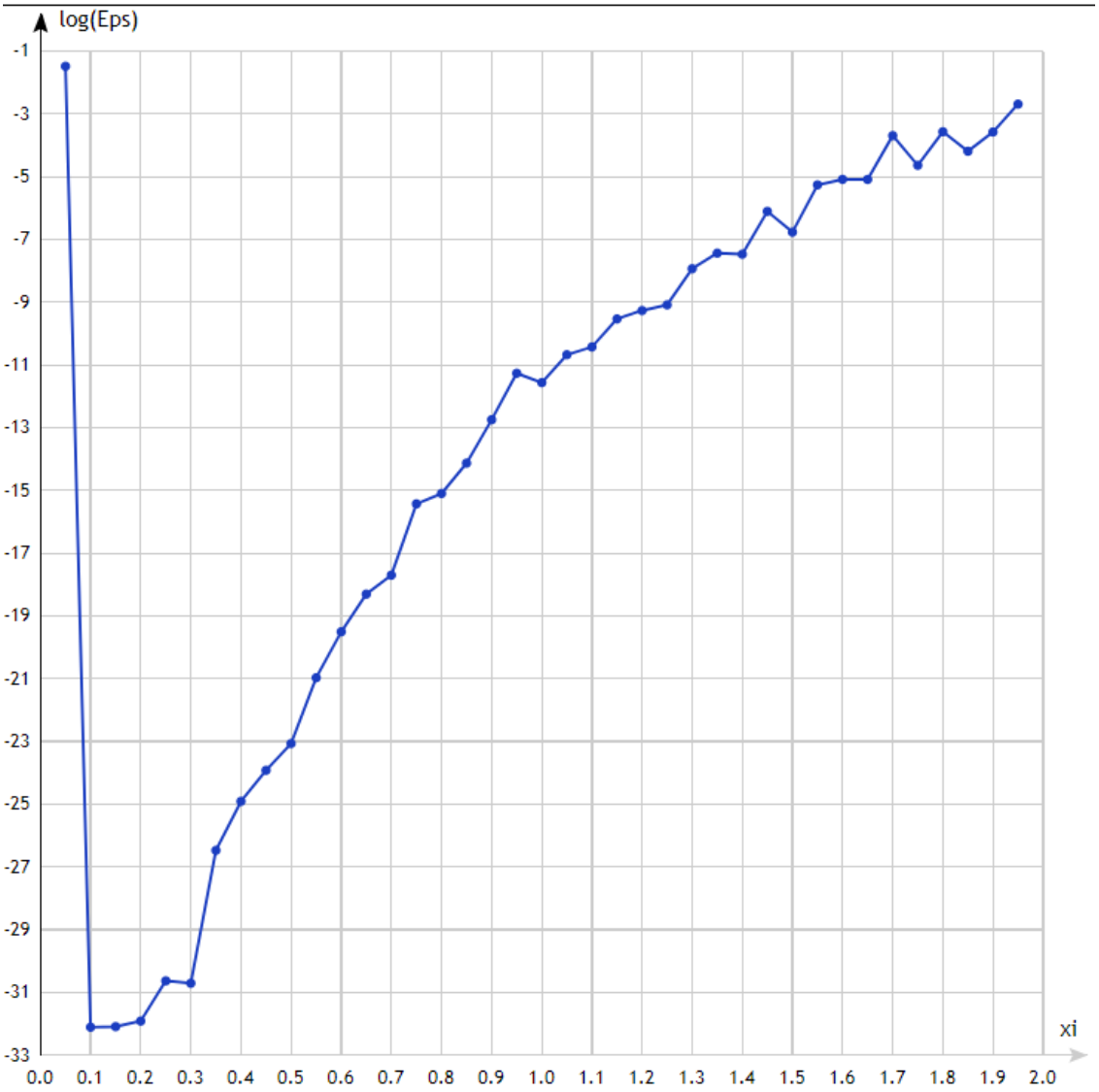

Рисунок 4.6 – Аналіз параметру xi для sphere.

Як видно з графіку (рис. 4.6) для нашого алгоритму при вище означених параметрах, найкращім значення xi для функції sphere лежать у проміжку (0.1; 0.2). При значеннях параметру xi менших за 0.1, швидкість збіжності різко зменшується. Як було зазначено вище, це пояснюється тим, що при малих значеннях xi, відхилення при формуванні нових розв'язків з вибраних раніше старих буде малим, а це означає те, що нові розв'язки будуть мало відрізнятися від старих, отже швидкість збіжності уповільниться. Така ж

залежність, очевидно, прослідковується і для інших функцій, проте оптимальні значення xi будуть більшими для інших функцій в порівнянні з sphere, оскільки вони мають локальні мінімуми і нам потрібна більша широта вибору для запобігання концентрації розв'язків у локальних мінімумах. Наприклад, для функції rosenbrock графік залежності похибки алгоритму від параметру xi (рис. 4.7) при ідентичних параметрах має наступний вид.

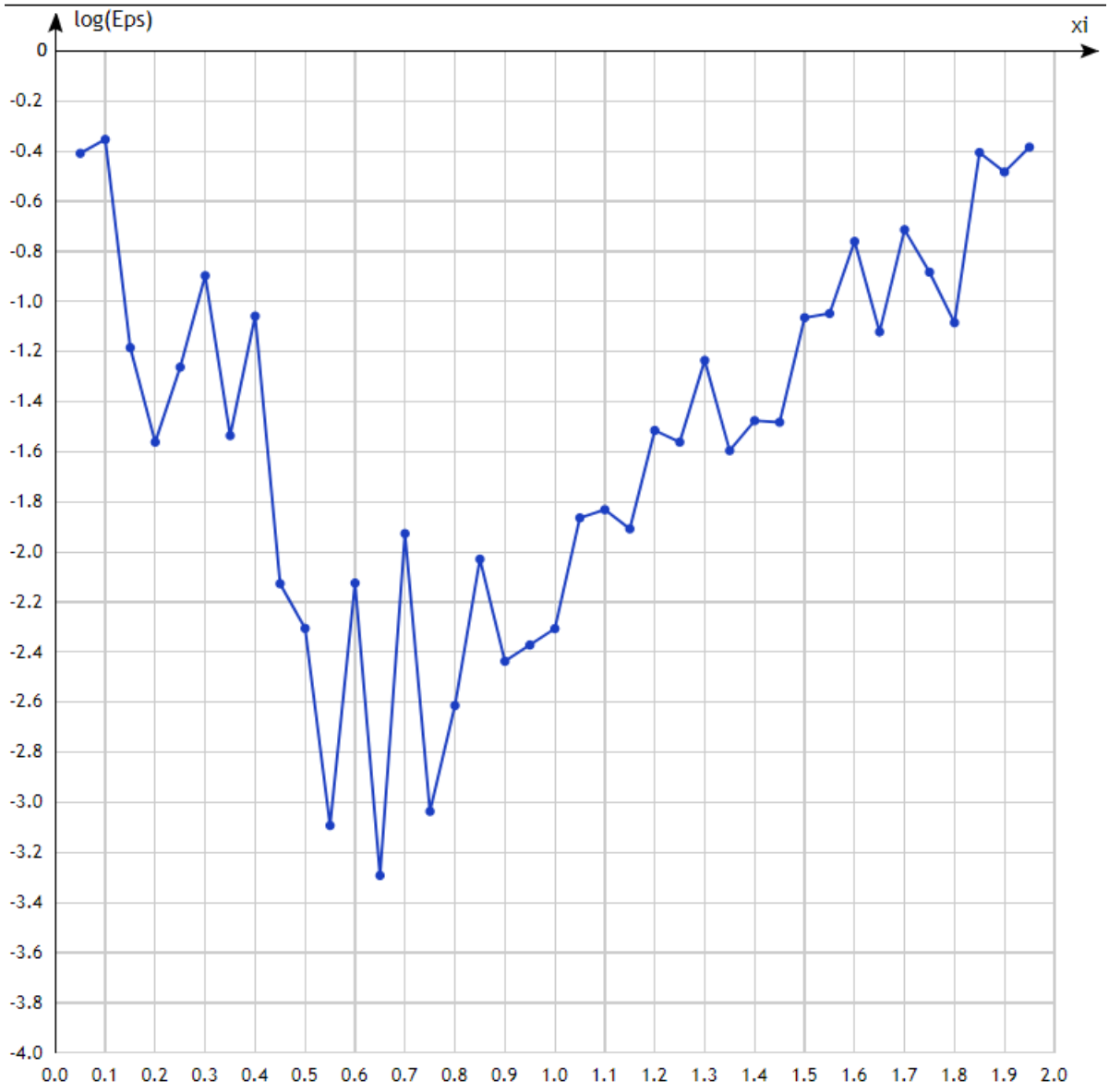

Рисунок 4.7 – Аналіз параметру xi для rosenbrock.

Наступним проаналізуємо, як впливає на роботу алгоритму параметр q. Для цього виконаємо тестування алгоритму при q = (0.1, 0.2, …, 1).

k = 10		xi = 1	k_del = 5	max_count = 100	
q	sphere	ackley	griewank	rastrigin	rosenbrock
0.1	5.63581e-20 (8.26702e-20)	1.17719e-09 (1.54438e-09)	0.0623097 (0.0268496)	0.995029 (0.994916)	1.52361 (2.43799)
0.2	2.22783e-17 (2.95616e-17)	1.98897e-09 (1.80511e-09)	0.0470372 (0.0233194)	0.882296 (0.886103)	0.131701 (0.124329)
0.3	5.55517e-15 (8.69343e-15)	9.05068e-09 (7.63923e-09)	0.0513181 (0.0327537)	0.270953 (0.339512)	0.211222 (0.234747)
0.4	2.76345e-14 (4.37324e-14)	3.20476e-08 (3.09017e-08)	0.0474389 (0.0284975)	0.466162 (0.642661)	0.0796156 (0.0948652)
0.5	1.96919e-13 (2.83268e-13)	1.71556e-07 (1.705e-07)	0.0479728 (0.0175666)	0.49085 (0.479165)	0.684424 (1.09735)
0.6	1.02392e-13 (1.30216e-13)	2.51494e-07 (1.98303e-07)	0.0598399 (0.0175419)	0.726687 (0.505611)	0.856219 (1.18079)
0.7	1.38196e-13 (1.884e-13)	2.6372e-06 (3.44311e-06)	0.0396161 (0.0256504)	0.912704 (0.370059)	0.103419 (0.0843413)
0.8	5.10664e-13 (6.87852e-13)	1.22647e-06 (9.71493e-07)	0.0564611 (0.0244504)	0.649514 (0.577164)	0.0769766 (0.103284)
0.9	2.25815e-13 (2.78439e-13)	5.61968e-06 (8.37619e-06)	0.0412251 (0.0142458)	1.07102 (0.525748)	0.123279 (0.137421)
1	3.36311e-10 (6.04417e-10)	2.23705e-05 (3.737e-05)	0.0328286 (0.0200972)	0.488155 (0.552716)	0.247951 (0.234459)

Таблиця 4.4 – Аналіз параметру q

N_alg = 5*100 + 10 = 510 для цього тестування.

З таблиці 4.4 видно, що для функції sphere, швидкість її збіжності обернено пропорційна параметру q. Це пояснюється тим, що параметр q відповідає за ймовірність вибору кращого розв’язку з набору, який у нас є на даний момент (чим менше його значення, тим більше ймовірність вибору кращого розв’язку) і функція sphere має лише один локальний мінімум, який і є глобальним. Тобто для мінімізації sphere нам потрібні лише найкращі розв’язки, оскільки нема ризику «застрягти» у локальному мінімуму, на відміну від інших функцій.

Розглянемо більш детально результати роботи алгоритму при значеннях q = (0.01, 0.02, …, 0.1).

k = 10		xi = 1	k_del = 5	max_count = 100	
q	sphere	ackley	griewank	rastrigin	rosenbrock
0.01	3.51106e-21 (5.12342e-21)	4.36412e-11 (5.42764e-11)	0.0327692 (0.0206484)	0.696471 (0.696471)	0.201765 (0.346422)
0.02	6.95459e-22 (1.08513e-21)	1.1212e-10 (1.77166e-10)	0.0629393 (0.0514924)	1.54123 (0.854491)	0.546313 (0.937383)
0.03	1.41836e-20 (2.26214e-20)	0.881315 (1.58637)	0.061422 (0.0556015)	1.09445 (0.537278)	0.405683 (0.636949)
0.04	4.40419e-20 (7.7943e-20)	4.75936e-11 (6.34218e-11)	0.0673516 (0.0304277)	0.49748 (0.49748)	0.30058 (0.411507)
0.05	2.83461e-21 (3.53126e-21)	4.35858e-11 (4.50295e-11)	0.035791 (0.026387)	0.795981 (0.795953)	0.947621 (1.46166)
0.06	8.73559e-20 (1.34064e-19)	0.257993 (0.464387)	0.059467 (0.0506754)	0.739227 (0.80156)	1.47231 (2.2613)
0.07	2.23969e-18 (3.98557e-18)	0.257993 (0.464387)	0.0432895 (0.0277233)	1.19395 (1.27355)	0.229473 (0.166552)
0.08	3.36451e-20 (4.93768e-20)	3.61401e-10 (5.78644e-10)	0.0498067 (0.0308712)	0.698518 (0.834127)	0.0588247 (0.0660007)
0.09	1.62556e-21 (2.03131e-21)	9.40119e-11 (6.36625e-11)	0.0723686 (0.0528792)	0.596983 (0.955157)	0.0902316 (0.078583)
0.1	4.78692e-20 (7.01238e-20)	0.257993 (0.464387)	0.0376795 (0.0185269)	1.09446 (0.756167)	0.184454 (0.207434)

Таблиця 4.5 – Аналіз параметру q

N_alg = 5*100 + 10 = 510 для цього тестування.

Для того, щоб швидкість збіжності функцій з локальними мінімуми мала таку ж залежність, необхідно збільшити вибірку наборів розв'язків k та кількість нових наборів на кожній ітерації k_alg. Розглянемо наступне тестування при k = 200 та k_del = 100.

k = 200		xi = 0.2	k_del = 100	max_count = 50	
q	sphere	ackley	griewank	rastrigin	rosenbrock
0.1	1.31618e-32	0	0.00263505	0	0.10076
0.2	8.63646e-25	7.67386e-13	0.00471914	0	2.70808e-05
0.3	4.45468e-19	4.21089e-10	0.00653823	0	0.0113194
0.4	2.25808e-14	2.02197e-08	0.00938075	8.97771e-12	0.000895601
0.5	1.37287e-12	4.66223e-07	0.00581916	1.36637e-11	0.0178661
0.6	1.32582e-12	1.83334e-06	0.00837823	2.56532e-09	0.00219819
0.7	1.28225e-10	3.04648e-06	0.00419448	3.42455e-10	0.00978309
0.8	2.85077e-11	1.22477e-06	0.00973098	5.30764e-08	0.000322913
0.9	2.19024e-11	1.2517e-05	0.00704258	7.97403e-10	0.00787829
1	6.76361e-11	3.43847e-05	0.00532815	3.73893e-09	0.00751725

Таблиця 4.6 – Аналіз параметру q

N_alg = 100*50 + 200 = 5200 для цього тестування.

Отже, як ми бачимо з таблиці 4.6, дійсно, при збільшенні k та k_alg, для більш швидкої мінімізації алгоритм потребує менші значення q.

Для функцій ackley та sphere, очевидно, найкращими будуть найменші значення q (таку ж залежність можна і віднести для rastrigin, але тільки при доволі великих k та k_del). Тепер же розглянемо, з якою швидкістю будуть мінімізовані функції при більших значеннях q.

k = 200		xi = 0.2	k_del = 100	max_count = 50	
q	sphere	ackley	griewank	rastrigin	rosenbrock
1	8.81598e-10	1.29102e-05	0.0072766	1.39963e-07	0.00214542
2	1.16404e-09	7.2456e-05	0.0190553	1.67106e-08	0.000480676
3	5.05217e-10	4.32402e-05	0.0244056	2.27246e-08	0.00452043
4	1.02378e-08	2.49304e-05	0.010316	5.25624e-07	0.0234511
5	1.57464e-09	7.28931e-05	0.0224379	3.73731e-08	0.00251026
6	7.67533e-09	1.77876e-05	0.0123628	1.70237e-07	0.00702193
7	8.14536e-09	3.69311e-05	0.00415487	9.43292e-10	0.00894478
8	6.40232e-09	1.45316e-05	0.0156192	2.01935e-05	0.053672
9	4.71238e-09	1.72427e-05	0.0241314	7.92005e-07	0.000796727

Таблиця 4.7 – Аналіз параметру q

N_alg = 100*50 + 200 = 5200 для цього тестування.

В таблиці 4.7 видно, що при збільшенні q до доволі великих значень, швидкість збіжності функцій практично не змінюється.

Побудуємо графік залежності похибки алгоритму від параметру q (рис. 4.8) для функції sphere. k = 100; xi = 0.2; k_del = 50; max_count = 50, а q змінюється від 0.05 до 2 з кроком 0.05 для цього тестування, тобто N_alg = 50*50 + 100 = 2100.

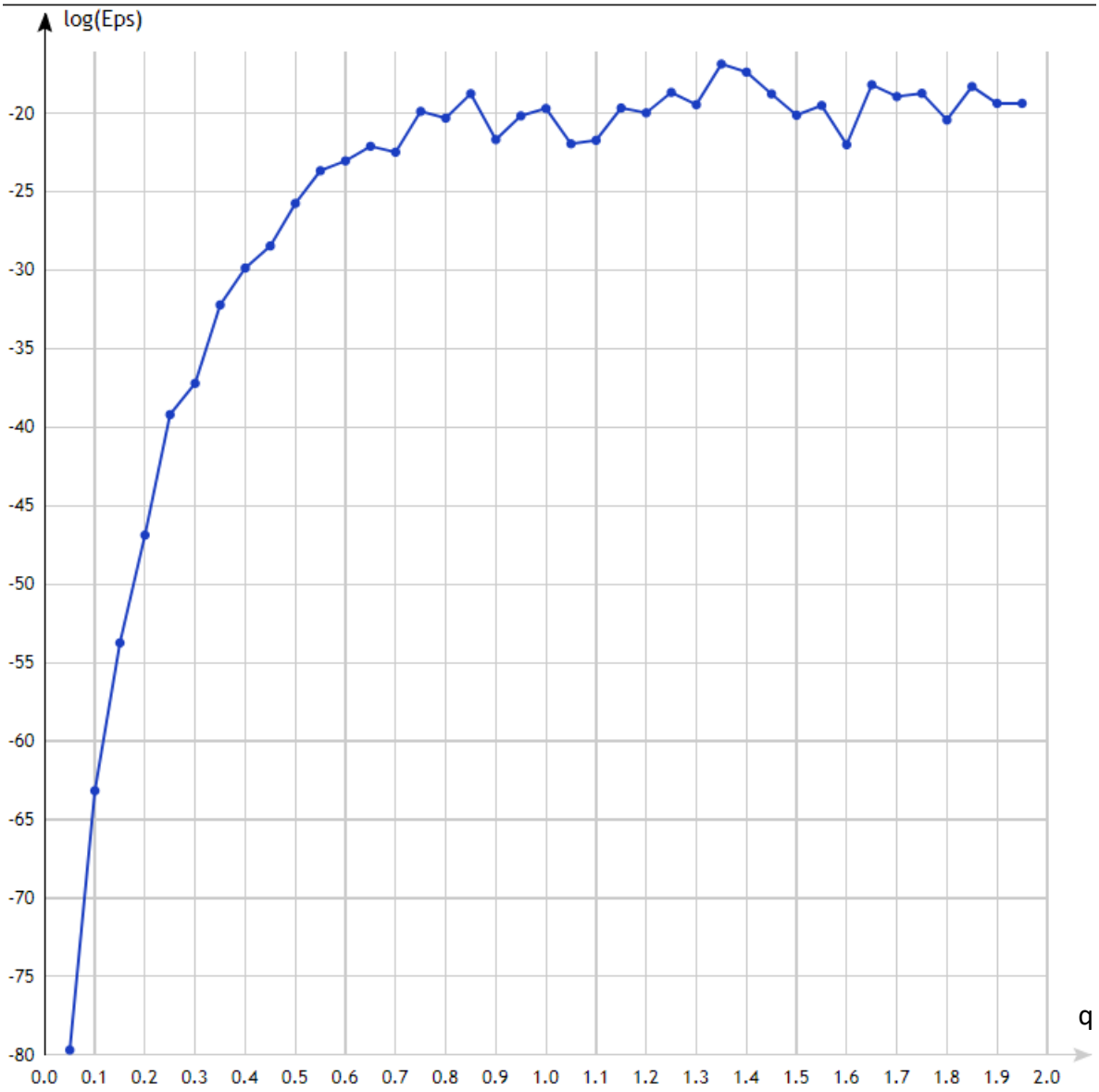

Рисунок 4.8 – Аналіз параметру q для sphere.

Як видно з графіку (рис. 4.8), найкращими є найменші значення q для функції, що поясняється тим, що sphere має лише один мінімум, який і є глобальним. Для інших функцій така залежність має місце лише при достатньо великих значеннях кількості наборів k та k_del. Проте навіть при k = 100 та k_del = 50 для функції ackley при поступовому зменшенні q, в деякий момент швидкість збіжності перестане збільшуватись, як видно на наступних графіках (рис. 4.9 та рис. 4.10). Для цих тестувань встановлено k = 100; xi = 0.2; k_del = 50; max_count = 10, а q змінюється від 0.001 до 0.01 з кроком

0.001, також від 0.01 до 0.05 з кроком 0.01, і ще від 0.05 до 2 з кроком 0.05, тобто N_alg = 50*10 + 100 = 600.

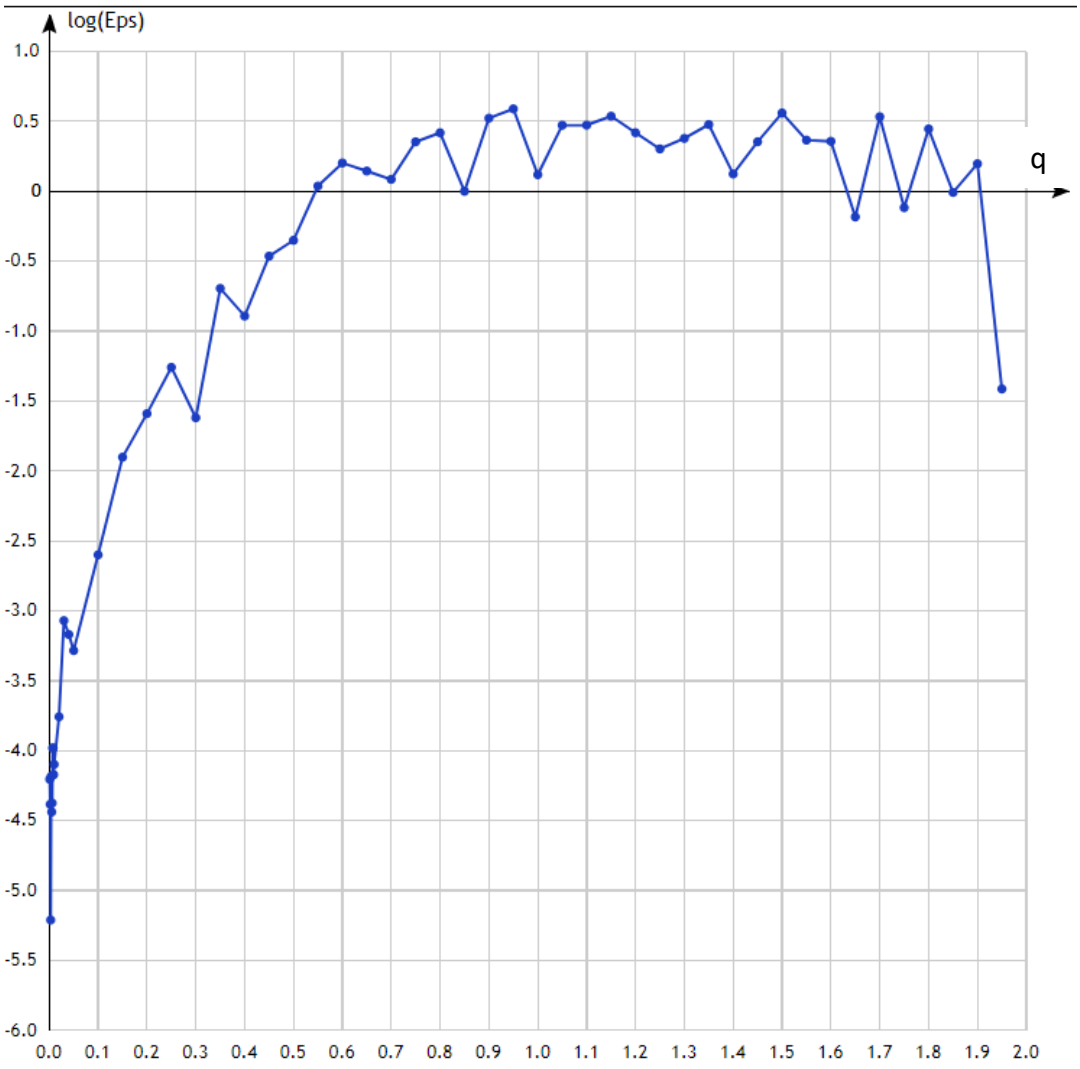

Рисунок 4.9 - Аналіз параметру q для ackley

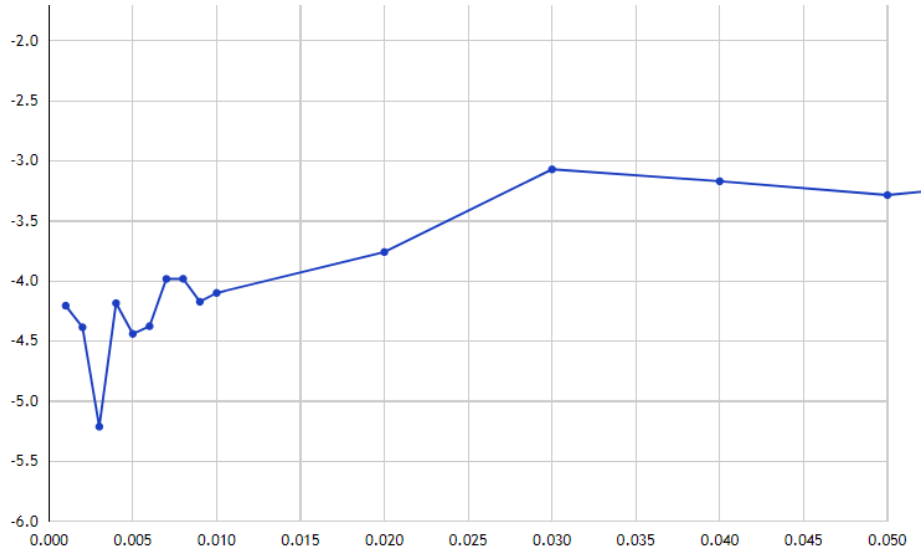

Рисунок 4.10 - Аналіз параметру q для ackley, графік масштабовано для менших значеннях q

Отже, для того щоб підібрати оптимальні значення параметрів q та xi треба враховувати значення k та k_del, а також характеристики цільової функції. Наприклад, можна виділити таку залежність: чим менше k та k_del, тим більшими є оптимальні значення xi або q. Це пояснюється тим, що якщо ми маємо маленькі набори, то для якісного формування нових розв'язків має бути відносно велике середньоквадратичне відхилення (за що відповідає параметр xi), щоб ці нові розв'язки були відмінні від старих, а також для функцій з локальними мінімуми важливо, щоб параметр не був занадто малим, адже в такому випадку достатньо того, щоб k розв'яків опинилися в локальному мінімумі і їх вихід з такого стану займе дуже багато часу.

Для знаходження оптимальної пари значень q та xi для функції sphere треба розглянути таблиці 4.8 і 4.9, для побудови яких (і відповідно всіх наступних, де проводиться аналіз для інших функцій) було зроблене наступне тестування: k = 40; k_del = 20; max_count = 50, а q змінюється від 0.05 до 1 з кроком 0.05, xi змінюється від 0.05 до 1 з кроком 0.05.

k = 40			k_del = 20	max_count = 50					
q xi	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45
0.05	8.25609	6.237	5.2904e-30	2.75193e-35	8.72101e-35	1.20716e-32	1.52131e-29	3.62678e-28	7.44298e-27
0.1	3.97216	3.17847	9.08211e-13	6.25033e-26	1.84783e-29	3.87954e-25	1.84599e-27	4.13734e-24	8.31652e-25
0.15	0.32135	0.00333955	5.67648e-11	8.20687e-18	1.4018e-23	2.59932e-22	1.16947e-22	2.49073e-18	1.51103e-20
0.2	1.21404	0.620969	0.0849036	5.05461e-20	2.5382e-20	2.5615e-22	4.2012e-19	3.05395e-18	3.52516e-20
0.25	4.5563	3.20783e-13	1.4028	3.00977e-19	2.074e-19	2.29072e-17	9.44246e-19	1.94077e-15	6.58538e-15
0.3	0.481137	0.000335425	6.61416e-08	9.40163e-16	1.18983e-14	3.59282e-16	2.57667e-15	1.85549e-14	1.81627e-15
0.35	2.80881	0.0861252	8.44952e-06	1.86878	1.21274e-14	1.96839e-14	1.38385e-12	3.15326e-14	1.50066e-13
0.4	0.156129	0.139682	6.237e-06	2.30299e-12	2.58475e-13	3.40708e-13	4.52641e-14	3.90575e-14	2.35752e-12
0.45	27.3196	0.0112717	1.69182e-11	9.18222e-12	7.14107e-14	7.98927e-13	1.87034e-11	6.00874e-13	7.07583e-14
0.5	0.00679475	1.03002	3.73664	0.00743161	5.5987e-12	7.7033e-11	6.66186e-12	1.41153e-12	5.38613e-13
0.55	3.51847	0.0908945	0.00157745	1.18921e-08	3.82646e-11	1.50986e-09	1.15665e-12	7.11055e-10	3.95067e-11
0.6	4.36099	37.7442	0.00124598	1.07037e-11	5.13768e-11	1.76526e-09	1.54703e-11	3.09536e-12	6.55615e-09
0.65	7.26504	0.715898	1.90896e-09	2.379e-10	5.98913e-09	1.43001e-10	4.50117e-09	2.01967e-10	7.61604e-11
0.7	6.16664	0.144897	1.69443	0.0589433	1.05376e-10	1.55826e-08	5.27462e-11	3.92836e-11	1.88965e-10
0.75	4.00032	0.131257	4.34747e-10	1.60738e-11	6.92093e-10	4.90112e-10	4.72022e-10	3.88276e-11	1.21246e-12
0.8	1.13098	0.0635677	0.000919803	2.9381e-09	4.03444e-09	9.338e-11	1.02334e-09	9.30733e-09	5.84159e-10
0.85	0.00279879	16.7838	4.43267e-12	2.60132e-09	2.98675e-09	2.16292e-10	4.61074e-10	1.43227e-09	8.90841e-10
0.9	1.62664	0.000164171	7.25925e-08	1.18172e-09	5.08583e-10	1.24789e-09	6.01361e-09	2.0578e-11	6.70956e-11
0.95	47.9955	0.000545344	1.02682	2.19301e-07	2.94168e-09	5.01221e-10	5.55809e-10	4.63331e-11	1.02911e-10

Таблиця 4.8 – Аналіз q та xi для sphere

0.45	0.5	0.55	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95
7.44298e-27	2.61551e-24	3.02976e-21	4.59561e-20	1.91004e-19	1.27372e-18	3.07334e-18	7.45192e-15	1.33502e-14	2.87157e-14	2.25898e-12
8.31652e-25	4.40234e-23	6.81442e-19	2.78627e-20	1.09066e-17	1.10143e-16	1.39535e-15	9.49986e-16	5.23012e-14	3.16303e-12	1.50131e-11
1.51103e-20	2.76239e-20	5.15432e-19	2.64137e-18	4.696e-16	7.654e-16	3.48718e-15	2.98316e-13	4.47135e-14	1.22428e-12	5.98593e-12
3.52516e-20	1.21692e-17	5.49373e-17	2.01183e-15	4.91974e-17	1.62156e-14	1.45109e-13	1.99082e-12	6.98568e-12	1.05391e-10	4.88094e-11
6.58538e-15	7.00017e-17	6.29595e-16	8.8383e-15	7.21426e-15	1.74962e-13	1.3062e-12	1.44648e-11	1.51494e-11	3.59502e-10	4.74998e-11
1.81627e-15	3.98801e-14	2.32886e-15	5.1282e-14	1.01134e-13	5.64612e-13	3.36554e-13	1.17813e-11	2.80826e-11	1.11875e-08	1.19381e-09
1.50066e-13	1.25778e-13	3.91205e-13	3.09743e-13	6.06313e-13	1.15434e-12	2.9638e-11	7.66543e-10	6.42228e-11	2.58898e-09	9.07801e-10
2.35752e-12	2.48177e-13	5.96328e-12	4.22362e-11	2.16901e-12	4.91983e-12	4.54773e-11	2.81129e-11	5.02285e-11	2.29799e-10	4.37971e-10
7.07583e-14	2.58268e-11	1.48151e-11	7.92883e-12	1.967e-11	1.0554e-10	2.73508e-10	6.42925e-09	2.64046e-09	8.61085e-10	1.4572e-08
5.38613e-13	2.27763e-13	1.27554e-12	3.96588e-11	9.25786e-12	3.46991e-10	2.53358e-10	1.25396e-10	1.10165e-09	8.6633e-09	9.97738e-09
3.95067e-11	2.19215e-12	1.15466e-09	2.12112e-10	2.67471e-10	1.14617e-10	3.57488e-10	3.76194e-08	6.86731e-09	8.18685e-08	8.77434e-09
6.55615e-09	4.90973e-10	4.14436e-11	3.177e-11	3.92059e-10	1.0032e-09	9.48791e-09	2.50398e-08	2.10123e-10	5.09479e-08	2.80976e-07
7.61604e-11	6.82476e-11	4.12289e-10	2.85071e-10	2.36075e-09	8.85668e-09	6.33883e-10	1.89267e-08	7.66681e-09	9.22934e-08	3.97016e-07
1.88965e-10	1.09853e-10	2.45245e-09	1.3685e-10	1.09402e-10	5.76604e-09	3.73782e-10	4.45817e-10	1.34249e-07	2.03751e-07	1.106e-08
1.21246e-12	3.58485e-10	5.60058e-12	9.97133e-10	1.12025e-10	9.67871e-10	2.39208e-10	2.24869e-09	6.66082e-08	2.20729e-08	6.24473e-08
5.84159e-10	5.9181e-11	8.68492e-09	3.58698e-11	2.24113e-10	1.99376e-09	5.14333e-09	9.21831e-09	1.40781e-08	8.61592e-08	6.65935e-08
8.90841e-10	2.1611e-11	1.76698e-10	3.46438e-10	8.68633e-10	3.86563e-11	1.06967e-08	3.34791e-09	1.33309e-07	1.81104e-08	9.89922e-07
6.70956e-11	1.84588e-09	7.06768e-10	7.0765e-12	7.03684e-09	1.32716e-08	1.35505e-08	9.23012e-08	6.23918e-09	3.41455e-07	8.13424e-07
1.02911e-10	1.42518e-10	3.62799e-10	1.24323e-08	1.41946e-09	1.81865e-09	2.37973e-08	1.48791e-07	2.53465e-08	1.66827e-07	2.43025e-07

Таблиця 4.9 – Аналіз q та xi для sphere

Отже, для функції sphere оптимальними для даного тестування є значення:

- q – 0.05;
- xi – 0.2;

Для знаходження оптимальної пари значень q та xi для функції ackley розглянимо таблиці 4.10 і 4.11.

q xi	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45
0.05	3.31613	2.87209	0.859976	3.54828e-05	1.18424e-15	0	2.36848e-15	3.07902e-14	2.14347e-13
0.1	5.06908	2.91623	8.90884e-08	3.25193e-10	1.90662e-13	1.29082e-13	3.83693e-13	2.27374e-13	9.33179e-12
0.15	3.21419	2.45134	0.00119655	0.000186777	9.20153e-13	2.22755e-12	4.5367e-11	2.5737e-11	1.21692e-10
0.2	3.87591	0.00762961	0.859976	9.98149e-05	2.85321e-09	2.08472e-10	1.42159e-10	2.28316e-10	2.03505e-09
0.25	3.20341	0.521729	0.00938277	2.18171e-06	1.2221e-05	9.88042e-10	8.53125e-09	4.6943e-09	1.45537e-08
0.3	3.42768	0.0835475	0.119694	3.32318e-05	1.25371e-08	3.11064e-08	2.08868e-08	2.61326e-08	3.34296e-07
0.35	3.15061	4.23346	0.87132	8.39364e-05	4.47423e-05	4.13462e-08	1.31517e-07	4.68048e-08	7.39019e-08
0.4	3.60238	1.73101	0.00242664	1.24362e-05	1.4406e-05	7.19205e-07	1.58965e-07	2.09272e-07	4.31719e-07
0.45	3.84273	1.79399	2.26595e-05	7.09113e-06	2.9661e-07	1.23257e-06	4.07728e-07	1.28608e-06	9.93402e-07
0.5	4.45108	0.864518	0.0380742	3.47293e-05	4.45009e-06	4.93708e-06	6.93514e-06	1.69214e-06	7.32289e-06
0.55	2.95813	2.456	0.0135462	7.90292e-06	4.37403e-06	2.95277e-05	1.71927e-05	3.69573e-06	3.5259e-06
0.6	1.24652	0.795809	0.860943	8.78293e-06	3.79399e-06	3.87225e-05	5.42515e-06	1.14373e-05	1.78359e-06
0.65	2.82789	0.0023117	8.16178e-05	0.00832084	1.52949e-05	3.40786e-05	1.6369e-05	9.34966e-06	1.96845e-05
0.7	2.60474	0.105807	0.00764578	0.000456697	8.09056e-05	1.75902e-05	2.21958e-05	4.41007e-06	5.89267e-06
0.75	5.1433	0.0772493	1.05688	0.000467734	4.12402e-05	7.89682e-05	1.50843e-05	1.58297e-05	2.6819e-05
0.8	4.11812	0.100879	4.0897e-05	5.1001e-05	6.63403e-05	5.35436e-05	4.40028e-05	7.35985e-06	2.34682e-05
0.85	2.77655	1.42231	0.0108336	0.000106056	9.50325e-05	3.52555e-05	1.81566e-05	2.14874e-05	2.92429e-05
0.9	0.140234	0.920244	0.008123	6.01502e-05	0.00013535	2.04117e-05	4.96977e-05	9.96432e-06	1.90838e-05
0.95	3.4322	0.0232097	0.000373047	0.00042882	3.09983e-05	0.000125215	5.83805e-05	2.20733e-05	2.02278e-05

Таблиця 4.10 – Аналіз q та xi для ackley

0.5	0.55	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95
5.36341e-12	2.7935e-11	5.24167e-11	8.99359e-10	1.50094e-09	7.84134e-09	7.90448e-08	1.92387e-07	3.36368e-07	4.10239e-07
3.75072e-11	9.27235e-11	5.11185e-10	2.31157e-09	3.5622e-09	1.95568e-08	1.05025e-07	2.22047e-07	3.41567e-07	1.8111e-06
4.04723e-10	4.52128e-10	2.74704e-09	3.2218e-08	6.93915e-08	8.25514e-08	3.6917e-07	1.03193e-06	3.20453e-06	4.85888e-06
3.53251e-09	3.12065e-08	3.89525e-08	2.10357e-07	9.14031e-08	3.1818e-07	2.4127e-07	1.93782e-06	1.06949e-05	1.18936e-05
2.11376e-08	4.40125e-08	4.73908e-08	9.05761e-07	2.60282e-07	1.69002e-06	2.9088e-06	8.29484e-06	1.89901e-05	2.78535e-05
1.03685e-07	6.65071e-08	3.44557e-07	6.5312e-07	2.14617e-06	2.24468e-06	1.25582e-05	1.9824e-05	7.31083e-05	9.60819e-05
1.97592e-07	3.26477e-07	2.9151e-07	3.85347e-06	1.85423e-06	1.19198e-05	3.11271e-05	1.9696e-05	9.24541e-05	0.000148588
1.50362e-06	1.06165e-06	5.31159e-06	6.92387e-06	8.11484e-06	2.36054e-05	2.30401e-05	3.21516e-05	6.24448e-05	8.4281e-05
2.4262e-06	1.64998e-06	3.27772e-06	1.03217e-05	2.82586e-05	3.21582e-05	5.98522e-05	5.87038e-05	6.65914e-05	0.000325519
5.19591e-06	2.06284e-06	2.41279e-05	8.2984e-06	1.09512e-05	1.64844e-05	2.29039e-05	0.000117632	0.000148269	0.000115683
2.75034e-06	1.30448e-05	5.32914e-06	1.26549e-05	3.26178e-05	9.14912e-05	5.21834e-05	7.11066e-05	9.20713e-05	0.00011024
1.15623e-05	1.71038e-05	2.52787e-06	9.88227e-06	2.63153e-05	8.00101e-05	0.000143441	0.000128171	0.000171919	0.000515558
1.54065e-05	4.16364e-05	2.29854e-05	1.79823e-05	4.31704e-05	2.90333e-05	4.7934e-05	0.000343905	0.000213183	0.000473083
1.4141e-05	9.46433e-06	3.78899e-05	2.70096e-05	3.50557e-05	2.37997e-05	7.78424e-05	0.000183118	0.000269674	0.000628922
1.1264e-05	3.45594e-05	2.24097e-05	4.18442e-05	0.000126056	0.000167494	0.000135297	0.000225446	0.000437577	0.000676036
2.25143e-05	0.000190918	1.89139e-05	1.30026e-05	0.000119914	4.96272e-05	4.25627e-05	0.000355492	0.000365942	0.000282798
1.2744e-05	5.66303e-05	3.76301e-05	6.86648e-05	0.000115447	6.74023e-05	0.00016294	0.000405411	0.000262216	0.000554702
1.4462e-05	4.5836e-05	4.45474e-05	1.8249e-05	7.44531e-05	0.000129923	0.00030719	0.000307579	0.000983615	0.000740845
6.86197e-05	6.13699e-05	3.11851e-05	8.84968e-05	7.42106e-05	0.000103923	9.63981e-05	0.000109864	0.00101503	0.000494686

Таблиця 4.11 – Аналіз q та xi для ackley

Отже, для функції ackley оптимальними для даного тестування є значення:

- q – 0.05;

- xi – 0.3;

Для знаходження оптимальної пари значень q та xi для функції griewank розглянимо таблиці 4.12 і 4.13.

q xi	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45
0.05	0.202749	0.237496	0.168504	0.161162	0.0361664	0.0238314	0.00712603	0.0241119	0.0416658
0.1	0.381682	0.0879304	0.020554	0.0427264	0.0213883	0.0117406	0.0320893	0.0393915	0.0239429
0.15	0.489956	0.0549244	0.299143	0.0394448	0.0221675	0.0180961	0.0177023	0.0186128	0.0155181
0.2	0.122569	0.0863147	0.05052	0.0168986	0.0143686	0.0278627	0.00866598	0.0260189	0.0435633
0.25	0.364045	0.131796	0.0403188	0.00743168	0.0411304	0.0239171	0.0141578	0.0204954	0.0269404
0.3	0.172009	0.0497162	0.0567323	0.0699272	0.0258816	0.0236192	0.038561	0.0149997	0.0283413
0.35	0.0616805	0.0493345	0.0981041	0.0458073	0.0572546	0.0183645	0.0126539	0.0184955	0.012702
0.4	0.24967	0.0656088	0.0673188	0.0601419	0.00935225	0.0262767	0.0161728	0.0398528	0.0104912
0.45	0.161274	0.104139	0.0714527	0.0470267	0.0273935	0.0215843	0.0217434	0.00826369	0.0315417
0.5	0.144825	0.36828	0.0317899	0.0276418	0.0541115	0.0174189	0.0234005	0.0316447	0.0275421
0.55	0.523658	0.0103782	0.0848565	0.0421232	0.010187	0.0136294	0.0197475	0.0288195	0.0205241
0.6	0.0486124	0.0205762	0.0389606	0.025322	0.0104086	0.0135971	0.0208331	0.0330367	0.0247132
0.65	0.510382	0.0625839	0.0246452	0.0373912	0.0323821	0.0379819	0.0236307	0.0464152	0.032937
0.7	0.332248	0.0381503	0.030526	0.0340401	0.0137567	0.046659	0.0443869	0.0198485	0.0158561
0.75	0.369766	0.0274143	0.0328153	0.0218811	0.0214492	0.0344799	0.0147416	0.0249267	0.0236691
0.8	0.213826	0.0391629	0.0142311	0.0249228	0.0212238	0.0464828	0.0190596	0.0409326	0.0248598
0.85	0.311445	0.0848099	0.0233136	0.0342715	0.0393255	0.0152044	0.0340935	0.0194966	0.0359502
0.9	0.0287773	0.00844188	0.0314923	0.0293387	0.0300228	0.0432357	0.0218617	0.0249489	0.036217
0.95	0.0525907	0.00416894	0.0481255	0.0221353	0.0152502	0.026608	0.0281462	0.0304178	0.0236791

Таблиця 4.12 – Аналіз q та xi для griewank

0.5	0.55	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95
0.0104603	0.0247686	0.0149724	0.0200919	0.0117326	0.0291989	0.0231596	0.0218633	0.0239198	0.0208645
0.0150775	0.0176625	0.0125798	0.028161	0.0154641	0.0244449	0.0153469	0.0241033	0.0209135	0.0280397
0.0146851	0.0240369	0.0188291	0.0227537	0.0106097	0.0399349	0.0516214	0.0170227	0.0625725	0.0276827
0.0137673	0.0306807	0.0212867	0.0127165	0.0334049	0.0217922	0.0232872	0.0286052	0.0271008	0.0191575
0.0180904	0.0194107	0.0415012	0.0197629	0.027021	0.022374	0.0159751	0.0123787	0.0428524	0.0424794
0.0157705	0.0222484	0.0301557	0.0166218	0.0302419	0.0296113	0.0267152	0.0369617	0.0213989	0.0258639
0.0318406	0.0245221	0.0263263	0.0198576	0.0275353	0.0296696	0.0357293	0.0359582	0.0605748	0.037935
0.0211718	0.0155935	0.0275109	0.0286094	0.0282449	0.03314	0.0268701	0.0265002	0.0205925	0.0332432
0.0236858	0.0162354	0.0270509	0.034317	0.0316561	0.0247148	0.0290533	0.0336761	0.0303719	0.0272733
0.0108574	0.0255174	0.02021	0.0346256	0.0181869	0.0423805	0.0402977	0.026131	0.0356526	0.0324032
0.0206834	0.0109352	0.0170685	0.0222038	0.0221314	0.0254957	0.0264898	0.0290027	0.0484849	0.0220085
0.0135675	0.0104843	0.012223	0.0220867	0.0315741	0.0183353	0.0353511	0.0166632	0.0511677	0.0536201
0.046315	0.0320963	0.0249235	0.0202803	0.0289916	0.0370278	0.032959	0.0284607	0.0263326	0.0508949
0.0316482	0.0363059	0.0181298	0.0227333	0.0521717	0.0273647	0.024493	0.0371914	0.0355743	0.0239266
0.0262091	0.0308834	0.0239592	0.0202129	0.0304153	0.0239123	0.0279861	0.0289481	0.0525797	0.0297379
0.0234031	0.0225885	0.042676	0.0141883	0.0171622	0.0428509	0.0432305	0.0325483	0.0443421	0.0332909
0.0262103	0.0146717	0.0268747	0.036082	0.0309211	0.0281832	0.033115	0.0467959	0.0357784	0.060236
0.0453414	0.0473547	0.0250156	0.0242756	0.026579	0.0362903	0.026251	0.0225288	0.0437923	0.0542631
0.0334723	0.0282816	0.0196428	0.0570884	0.0261637	0.0314458	0.038093	0.0225865	0.0380306	0.0220435

Таблиця 4.13 – Аналіз q та xi для griewank

Отже, для функції griewank оптимальними для даного тестування є значення:

- q – 0.95;
- xi – 0.1;

Для знаходження оптимальної пари значень q та xi для функції rastrigin розглянимо таблиці 4.14 і 4.15.

q xi	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45
0.05	0.664281	1.98992	0.995109	1.65827	1.65827	0.994959	1.65826	0.994959	0.331653
0.1	1.32668	0.334863	1.32661	0.663306	0.663306	0.994959	0.331653	0.331653	0.331653
0.15	0.334878	0.332448	2.32165	0.663306	0.663306	0.331653	0.331653	0.663306	0.663306
0.2	0.00118941	0.663385	0.995666	0.331655	1.36187e-14	5.23931e-11	0.00543364	4.61705e-11	2.38041e-10
0.25	0.99834	0.331675	0.99694	0.331653	7.40918e-12	9.12109e-08	5.38828e-14	5.86752e-09	0.0425408
0.3	0.996055	0.352965	0.663306	4.31298e-09	1.85511e-12	7.22631e-07	2.11828e-07	6.70523e-08	0.000881561
0.35	0.996194	0.663386	0.331653	2.04873e-12	0.994959	1.03611e-08	0.331777	0.741013	6.13329e-05
0.4	0.663739	0.00120222	0.000282738	3.44592e-06	6.46982e-09	0.331653	0.331653	0.0335471	0.0103652
0.45	0.998999	0.995775	0.331653	0.331653	3.1697e-07	0.00174627	2.49298e-08	0.0038903	0.00262895
0.5	0.663755	0.663318	0.331653	3.95495e-07	1.67673e-07	0.331653	2.49041e-06	1.98506e-05	0.000293493
0.55	0.378258	0.332528	0.331653	0.331679	0.00271442	5.52769e-05	0.331939	0.0178982	0.331662
0.6	0.000204564	0.995817	0.331731	7.86506e-07	4.08651e-07	5.8873e-07	0.424195	0.000845935	0.00362214
0.65	1.00116	0.00052705	0.331677	0.663309	1.70329e-09	0.6637	0.0225885	0.00436365	0.228185
0.7	0.332005	0.664251	0.000254477	0.331658	2.69201e-07	0.00213478	0.00186324	0.0292923	0.000304587
0.75	1.33169	0.331763	8.15233e-05	0.331655	0.331695	0.000588222	1.29837e-06	0.014272	0.0111052
0.8	0.665544	0.00025565	0.663872	0.663321	0.393459	2.14293e-05	0.000444631	0.332723	0.33459
0.85	0.665439	0.332585	6.35328e-05	2.53603e-05	1.4008e-07	0.00226644	7.53236e-05	0.00724666	0.00372002
0.9	3.98109e-05	1.15405e-05	0.000291567	0.00799694	0.331655	0.426679	2.78062e-05	0.0690865	0.351145
0.95	0.663767	0.333248	1.01483e-07	5.7e-05	1.45722e-05	5.48466e-05	6.92496e-06	0.000384334	0.0065853

Таблиця 4.14 – Аналіз q та xi для rastrigin

0.5	0.55	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95
0.331653	0.663306	2.92507e-13	0.00133425	0.00181158	0.000142868	0.0597722	0.229344	0.0653797	0.448816
0.331653	0.180558	0.339978	0.217769	0.464528	0.331653	6.44371e-06	0.31631	0.687469	0.354077
0.663306	0.510744	0.103468	0.0290591	0.0629288	0.355525	0.336764	0.198139	0.176004	0.217208
0.0214696	0.331653	0.0687775	0.000307476	0.301714	0.0150714	0.335529	0.382578	0.160205	0.57002
1.18197e-08	0.331653	0.0392214	0.000289219	0.0775273	0.0322778	0.0838753	0.0678175	0.338436	0.278484
0.331653	4.13201e-07	0.0397877	0.0337966	0.593882	0.344215	0.0434878	0.269409	0.0623225	0.819287
5.8442e-09	0.00761135	0.00037378	0.000423355	0.0663243	0.270089	0.214815	0.159525	0.567615	0.245903
0.0759763	0.130425	0.334592	0.0324868	0.48387	0.299715	0.126042	0.36972	0.159717	0.281769
0.332012	0.00372644	0.334539	0.240183	0.170657	0.521869	0.328449	0.293202	0.110319	0.0975329
0.30792	0.642995	0.24203	0.0451815	0.205406	0.259491	0.414285	0.392443	0.138272	0.775453
0.432563	0.0163574	0.0369972	0.125253	0.0568084	0.457685	0.602279	0.788005	0.236985	0.58194
0.00320527	0.0567337	0.00204694	0.395814	0.430452	0.550146	0.32289	0.485923	0.677067	0.653153
0.148257	0.0213109	0.312285	0.786153	0.438264	0.309299	0.312355	0.574506	0.190482	0.425649
0.0229437	0.245916	0.3547	0.332243	0.268344	0.0563131	0.167113	0.381197	0.566985	0.422387
0.0837625	0.334209	0.331728	0.0580106	0.0867989	0.412154	0.0337651	0.263209	0.892736	0.855719
0.0505402	0.0940647	0.0366509	0.491105	0.0341927	0.545593	0.566623	0.25268	0.423688	0.889697
0.0998115	0.332426	0.0334971	0.0191889	0.428775	0.68428	0.56769	0.607841	0.0495906	0.127108
0.126035	0.154737	0.43463	0.14449	0.149516	0.18246	0.210688	0.634422	0.67656	0.720458
0.10435	0.100173	0.204732	0.0620678	0.476115	0.430478	0.258606	0.151959	0.581924	0.504594

Таблиця 4.15 – Аналіз q та xi для rastrigin

Отже, для функції rastrigin оптимальними для даного тестування є значення:

- q – 0.25;
- xi – 0.2;

Для знаходження оптимальної пари значень q та xi для функції rosenbrock розглянимо таблиці 4.16 і 4.17.

q xi	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45
0.05	0.969558	1.04448	0.812431	0.279804	0.25591	1.31856	0.419898	0.155667	0.16828
0.1	1.51331	2.72477	0.0619472	0.614478	1.46612	0.236301	3.46987	0.00382047	0.185553
0.15	0.540947	1.33522	0.0377897	0.155072	0.00949914	0.0190393	2.4961	0.489755	0.0135137
0.2	3.63891	0.63701	0.190586	0.367133	0.143078	0.669699	0.0953599	0.0275522	0.0148732
0.25	0.618832	0.00392718	0.0993	0.180191	0.0232423	0.201195	0.0755667	0.124815	0.152653
0.3	0.253932	0.17734	0.0878802	0.141745	0.634876	0.0764423	0.0564119	0.118284	0.117192
0.35	1.42971	1.04836	0.108465	0.139365	1.36444	0.675156	0.420715	0.890493	0.194958
0.4	0.593236	2.9083	0.658708	0.110301	0.0018468	0.0435879	0.0659216	0.295197	0.0127112
0.45	0.221386	0.926756	0.221762	0.141263	0.146593	0.166658	0.156531	0.214257	0.061576
0.5	1.64315	0.555404	0.306522	0.271339	0.0508987	0.845933	0.181649	0.0786807	0.0678797
0.55	1.12827	0.160188	0.410164	0.0778455	0.0666772	0.0887209	0.00412913	0.00992242	0.0255072
0.6	0.328708	0.195615	0.10705	0.248864	0.932683	0.0537296	0.4869	0.104463	0.0191392
0.65	0.263654	0.301811	0.229737	0.14023	0.252102	0.18099	0.0133639	0.0602835	0.0465977
0.7	0.150238	0.0345454	0.0890689	0.254369	0.124119	0.0931805	0.184243	0.193996	0.0862425
0.75	0.720598	0.15663	0.0360682	0.372796	0.0928213	0.0728879	0.0454484	0.00267101	0.0851201
0.8	0.0395284	0.199409	0.17312	0.0827916	0.0790021	0.114207	0.0149242	0.350085	0.0707077
0.85	0.548567	0.220044	0.220619	0.18714	0.24116	0.0599909	0.00801552	0.0651495	0.128957
0.9	0.845104	1.00071	0.091241	0.774185	0.0702029	0.0896055	0.332436	0.153039	0.0443725
0.95	1.06882	0.39966	2.5616	0.0722497	0.0494693	0.00925577	0.0762093	0.0646124	0.495279

Таблиця 4.16 – Аналіз q та xi для rosenbrock

0.5	0.55	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95
0.230406	0.0104566	0.00927464	0.0545688	0.00145158	0.00798392	0.0248401	0.00737062	0.00132218	0.00215268
0.137706	0.183479	0.00326874	0.0374008	0.00306282	0.00827879	0.0313459	0.00111479	0.084449	0.0289993
0.00187857	0.0935045	0.0741126	0.0261418	0.0074322	0.0445511	0.00602156	0.0104738	0.661872	0.0363541
0.0578701	0.148964	0.0201964	0.00276764	0.00583873	0.00258188	0.00151926	0.0157587	0.0085868	0.0110383
0.211299	0.0721031	0.0414282	0.0856431	0.00640331	0.0592301	0.0352518	0.0192681	0.0121641	0.0225997
0.0273861	0.0865617	0.816564	0.0293325	0.0255818	0.127499	0.00152874	0.0573885	0.0356409	0.0327294
0.0246666	0.104851	0.00534457	0.0359661	0.0848977	0.176	0.0581472	0.15606	0.0926543	0.0202942
0.0516812	0.0849715	0.114365	0.0797642	0.154214	0.0568036	0.0520828	0.0158389	0.208467	0.0768872
0.0759564	0.0153359	0.0384787	0.0773771	0.105346	0.048369	0.264222	0.107454	0.0550468	0.130003
0.0812888	0.070638	0.0632013	0.0230888	0.0274072	0.0848091	0.0435474	0.0472666	0.0556765	0.0132483
0.091991	0.177165	0.0577211	0.0966619	0.0348384	0.00825137	0.0722777	0.0274091	0.0354647	0.152957
0.124552	0.028531	0.0627047	0.0865851	0.0402987	0.0611729	0.0649515	0.0475018	0.0373857	0.0924291
0.0242309	0.252453	0.150362	0.0741237	0.232487	0.188639	0.0303761	0.044999	0.034595	0.0570358
0.0399766	0.0660768	0.0491485	0.0777577	0.0418493	0.0418228	0.0118579	0.148384	0.0545357	0.128379
0.102698	0.05123	0.131019	0.109428	0.0667539	0.259134	0.0124801	0.0624993	0.0209469	0.0660366
0.0187814	0.0957647	0.00757806	0.0384387	0.0767643	0.0255503	0.0162019	0.0157144	0.26002	0.0701457
0.0401219	0.111527	0.0580069	0.0198031	0.0276893	0.0708297	0.0354849	0.0450966	0.043104	0.03364
0.156751	0.0327212	0.120721	0.0507849	0.0635569	0.109741	0.0499882	0.106432	0.0612312	0.100726
0.0488304	0.17873	0.0880716	0.019245	0.0322757	0.655898	0.0468596	0.117775	0.0690693	0.073652

Таблиця 4.17 – Аналіз q та xi для rosenbrock

Отже, для функції rosenbrock оптимальними для даного тестування є значення:

- q – 0.1;
- xi – 0.85;

Проаналізуємо яке відношення параметрів k/k_del є найкращім. Для цього протестуємо алгоритм для таких значень: k = 30, k_del = (5, 10, 15, 25, 30, 45, 60, 75, 90).

k = 30	q = 0.1	xi = 1	max_count = 100		
k/k_del	sphere	ackley	griewank	rastrigin	rosenbrock
6	208.938	8.61462	9.36288	73.6961	858.34
3	11.3345	4.54345	1.05554	61.0249	44.6071
2	0.362337	0.301183	0.905454	51.7513	10.3096
1.5	0.0569493	0.293043	0.840027	60.4895	9.5811
1.2	0.00271761	0.0276795	0.464225	40.8043	8.0088
1	0.000220321	0.0119034	0.495159	59.4199	6.77249
0.66	3.81822e-06	0.00110941	0.694434	43.9048	6.91488
0.5	4.91896e-09	2.39954e-05	0.244806	50.0685	6.54121
0.4	9.0734e-11	3.67562e-06	0.637012	43.2335	6.54484
0.33	1.15518e-11	1.7891e-06	0.670347	37.2499	6.39116

Таблиця 4.18 – Аналіз параметрів k та k_del

З таблиці 4.18 можна побачити, що швидкість збіжності функцій sphere і ackley є пропорційною k_del, проте для функцій griewank, rastrigin і rodenbrock швикість досягнення мінімумів практично не змінюється при збільшенні k_del, тобто при k/k_del<1 для даного дослідження. Для наступного дослідження зафіксуємо значення параметру k_del = 20, а k будемо змінювати в інтервалі (5, 80).

k = 20	q = 0.1	xi = 1	k_del = 20	nax_count = 200	
k	sphere	ackley	griewank	rastrigin	rosenbrock
5	23.9112	3.09341	0.197012	7.19585	199.389
10	7.61549e-44	2.4869e-15	0.123958	3.58185	1.4386
20	2.10502e-30	3.55271e-15	0.207768	3.69834	1.52819
40	1.69666e-18	9.67703e-10	0.230723	7.55783	1.3685
80	3.8375e-09	3.15603e-05	0.283176	8.70874	1.75937

Таблиця 4.19 – Аналіз параметрів k та k_del

З таблиці 4.19 можна побачити, що найкращім співвідношення k/k_del для функцій sphere, griewank і rastrigin є 20/10 = 2/1. Для функції ackley – 20/20 = 1, для rosenbrock – 20/40 = ½.

4.2. Порівняльний аналіз

Порівняння будемо проводити між трьома алгоритмами:

- генетичний алгоритм (в таблицях позначено як GA);
- класичний мурашиний алгоритм оптимізації для неперервного простору (в таблицях ACO);
- мурашиний алгоритм оптимізації з метафорою агрегацією феромонів (в таблицях ACO*).

Порівняльний аналіз можна проводити такими способами:

1. Значення N_alg – кількість обчислень функції, повинне бути однаковим для усіх алгоритмів. Оцінюється середня похибка алгоритму за 50 тестів, чим менше вона, тим краще алгоритм. Також оцінюється середнє відхилення похибки.
2. Постійною є похибка eps. Оцінюється кількість обчислень функції.

Порівняємо алгоритми згідно з способом 1. Нехай N_alg = 100000.

Отже, нехай k_del = 100 і max_count = 1000 для нашого алгоритму. Визначимо оптимальні параметри xi, q та k нашого алгоритму для кожної з тестових функцій згідно з методикою, наведеною в попередньому пункті.

Для sphere:

- k = 200;
- k_del = 100;
- max_count = 1000;
- xi = 0.1;
- q = 0.01;

Для ackley:

- k = 100;
- k_del = 100;
- max_count = 1000;
- xi = 0.1;
- q = 0.1;

Для griewank:

- k = 200;
- k_del = 100;
- max_count = 1000;
- xi = 0.1;
- q = 0.05;

Для rastrigin:

- k = 200;
- k_del = 100;
- max_count = 1000;
- xi = 0.2;
- q = 0.05;

Для rosenbrock:

- k = 50;
- k_del = 100;
- max_count = 1000;
- xi = 0.2;
- q = 0.6;

Отже, визначивши найкращі параметри для нашого алгоритму для кожної з тестових функцій, можна починати порівняльний аналіз. Результати такого аналізу викладені нижче для значень n (кількість аргументів функцій): 10, 20, 30, 40.

	n = 10		
	GA	ACO	ACO*
sphere	5.31886e-19 (6.26508e-18)	8.27866e-24 (2.73974e-24)	6.07227e-28 (1.01126e-28)
ackley	8.73049e-08 (2.56697e-07)	7.71944e-12 (8.40556e-11)	9.06366e-13 (4.15262e-13)
griewank	4.28007e-02 (1.82263e-01)	1.13486e-04 (3.73595e-03)	5.62181e-04 (7.39451e-04)
rastrigin	1.04202e-06 (2.07175e-05)	5.01123e-11 (9.89673e-11)	2.49309e-12 (2.32582e-12)
rosenbrock	6.46449e-05 (6.41588e-05)	6.14066e-05 (5.78497e-05)	4.40092e-05 (1.0802e-06)
	n = 20		
	GA	ACO	ACO*
sphere	6.46916e-12 (1.51335e-11)	4.1691e-12 (5.0571e-13)	6.96741e-14 (2.49611e-14)
ackley	7.54009e-05 (8.88375e-05)	3.71755e-10 (5.67895e-10)	8.22346e-11 (8.04987e-11)
griewank	3.11795e-01 (5.15873e-01)	2.28104e-03 (5.85418e-03)	9.33146e-03 (9.60311e-02)
rastrigin	9.99725e-04 (9.97116e-03)	8.59288e-08 (8.01691e-09)	2.88641e-08 (8.76016e-08)
rosenbrock	1.21369e-04 (8.5613e-03)	3.67553-04 (3.39592e-05)	4.53194e-04 (6.50349e-04)
	n = 30		
	GA	ACO	ACO*
sphere	9.26911e-05 (1.0791e-05)	1.50594e-08 (7.09485e-08)	1.83361e-10 (4.38279e-10)
ackley	7.53844e-03 (8.5385e-03)	7.22065e-08 (6.29118e-08)	3.45607e-08 (3.48299e-08)
griewank	9.54241e-00 (5.12165e-01)	5.21613e-02 (7.69362e-02)	2.84823e-02 (5.36445e-02)
rastrigin	6.4807e-03 (7.61507e-02)	4.46712e-06 (6.39143e-06)	1.97452e-06 (7.69994e-06)
rosenbrock	8.22593e-02 (4.82638e-03)	3.02594e-02 (2.36399e-02)	4.25205e-02 (6.15165e-02)
	n = 40		
	GA	ACO	ACO*
sphere	6.91577e-03 (2.52083e-03)	4.11005e-04 (7.76394e-04)	9.90991e-06 (5.65395e-07)
ackley	9.54625e-01 (2.32746-01)	5.54326e-06 (7.29783e-06)	1.57186e-06 (5.42708e-06)
griewank	1.63338+01 (4.83901+01)	3.72744e-01 (7.23603e-01)	9.14606e-01 (2.27418e-01)
rastrigin	6.23844-02 (8.39512-02)	8.89529e-04 (2.3791e-04)	7.14869e-03 (9.69951e-03)
rosenbrock	5.53529-01 (2.39998-01)	5.27738e-01 (8.35475e-01)	2.60103e-01 (2.72216e-01)

Таблиця 4.20 - Порівняльний аналіз алгоритмів.

Отже, з таблиці 4.20 видно, що розроблений мурашиний алгоритм оптимізації з метафорою агрегації феромонів показує найкращі результати практично для всіх функцій. Середнє стандартне відхилення є достатньо малим, що свідчить про сталість алгоритму.

ВИСНОВКИ

Метою даної дипломної роботи була розробка модифікації мурашиного алгоритму оптимізації в неперервному просторі у вигляді системи агрегації феромонів з метою покращення точності й сталості результатів для рішення задачі оптимізації.

В результаті аналізу існуючих рішень був обраний саме мурашиний алгоритм як найперспективніший з евристичних алгоритмів оптимізації й запропонована його модифікація у вигляді мурашиного алгоритму оптимізації з метафорою агрегації феромонів. Головною особливістю даного алгоритму є використання так званих агрегаційних феромонів.

Для розробки програмної моделі цього алгоритму була використана мова програмування C++.

Розроблена програма дозволяє:

- задавати параметри алгоритму;
- задавати цільову функцію, яку потрібно мінімізувати;
- отримувати порівняльні таблиці для різних значень параметрів;
- отримувати порівняльні таблиці для різних цільових функцій;
- отримувати порівняльні таблиці для різних евристичних алгоритмів оптимізації;

За допомогою розробленої програми був проведений аналіз параметрів алгоритму, визначені їх оптимальні значення для кожної тестової функції. Показники роботи мурашиного алгоритму оптимізації з метафорою агрегації феромонів були порівняні з іншими евристичними алгоритмами реалізованими для задачі оптимізації. Отримані дані показують, що розроблений алгоритм практично для усіх проведених тестувань показує

кращі результати, ніж інші. Це дозволяє стверджувати, що даний алгоритм є доволі ефективним для рішення задачі оптимізації.

Розроблений алгоритм також може бути використаний для розв’язання прикладних задач оптимізації, таких як задача комівояжера, задача маршрутизації транспорту та телекомунікаційних мереж, а також різних задач з області логістики.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Dorigo, M., Stutzle, T., Ant Colony Optimization. MIT Press, Cambridge, MA, 2004.
2. Bell, W. J.: Chemo-orientation in Walking Insects, Chemical Ecology of Insects, Bell, W. J. and Carde, R. T. Eds, 93-109 (1984).
3. Bullnheimer, B., Hartl, R. F., and Strauss, C.: A New Rank Based Version of the Ant System: A Computational Study, Central European Journal for Operations Research and Economics, 7(1):25-38 (1999).
4. Bullnheimer, B., Hartl, R. F., and Strauss, C.: Applying the Ant System to the Vehicle Routing Problem, Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization, Voss, S., et al (Eds.), Kluwer (1999).
5. Costa, D. and Hertz, A.: Ants Can Colour Graphs. Journal of the Operational Research Society, 48:295-305 (1997).
6. Dorigo M., Maniezzo, V., and Colorni, A.: The Ant System: Optimization by a Colony of Cooperating Agents, IEEE Trans. on SMC-Part B, 26(1):29-41(1996).
7. Dorigo M. & L.M. Gambardella (1997). Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. IEEE TEC, 1(1):53-66 (1997).
8. Forsyth P. and Wren, A.: An Ant System for Bus Driver Scheduling. Proc. of the 7th Int. Workshop on Computer-Aided Scheduling of Public Transport (1997).
9. Goldberg, D. E.: Genetic algorithms in search, optimization and machine learning. Reading, MA: Addison-Wesley publishing company (1989).
10. Higuchi, T., Tsutsui, S., and Yamamura, M.:Theoretical analysis of simplex crossover for real- coded Genetic Algorithms, Proc. of the PPSN VI, 365-374 (2000).

11. Kuntz P., Layzell, P., and Snyers, D.: A Colony of Ant-like Agents for Partitioning in VLSI Technology. Proc. of the 4th European Conf. on Artificial Life, 417-424, MIT Press (1997).
12. Larranaga, P. and Lozano, J. A. (eds): Estimation of distribution algorithms, Kluwer Academic Publishers (2002).
13. Law, J. H. and Regnier, F. E.: Annual Review of Biochemistry, 40:533-548 (1971).
14. Lorenzo Figueiras, A.N. and Lazzari, C. R.: Aggregation behavior and interspecific responses in three species of Triatomine, Memórias do Instituto Oswaldo Cruz . 93(1):133-137 (1998).
15. Ono, I. and Kobayashi, S.: A Real-Coded Genetic Algorithm for Function Optimization Using the Unimodal Normal Distribution Crossover, Proc. of the 7th ICGA, 246-253, (1997).
16. Pelikan, M., Goldberg, D. E., and Lobo, F.: A survey of optimization by building and using probabilistic models. Computational Optimization and Applications, 21(1), 5-20 (2002). Also Technical Report IlliGAL Report 99018, University of Illinois at Urbana-Champaign (1999).
17. Pena, J. M., Lozano, J. A., and Larranaga, P.: Benefit of Data Clustering in Multimodal Function Optimization via EDAs, Estimation of Distribution Algorithms, Larranaga, P. and Lozano, J. A. (Eds), Kluwer Academic Publishers, Chap. 10:211-229 (2002).
18. Stützle, T. and H. Hoos, H.: The MAX-MIN Ant System and local Search for Combinatorial Optimization Problems: Towards Adaptive Tools for Global Optimization, Proc. of the 2nd Metaheuristics Int. Conf. (MIC-97) (1997).
19. Schatzman, M., Taylor, J., and Schatzman, M.: Numerical Analysis: A Mathematical Introduction, Oxford Univ Press (2002).